NASA/TP—2001-211331

System Simulation by Recursive Feedback: Coupling a Set of Stand-Alone Subsystem Simulations

D.D. Nixon
Marshall Space Flight Center, Marshall Space Flight Center, Alabama

National Aeronautics and
Space Administration

Marshall Space Flight Center • MSFC, Alabama 35812

October 2001

TRADEMARKS

Trade names are used in this report for identification purposes only. This usage does not constitute an official endorsement, either expressed or implied, by the National Aeronautics and Space Administration.

Available from:

NASA Center for AeroSpace Information
7121 Standard Drive
Hanover, MD 21076–1320
(301) 621–0390

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
(703) 487–4650

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF ACRONYMS

MARSYAS	Marshall System for Aerospace Simulation
MSFC	Marshall Space Flight Center
ODE	ordinary differential equation
RF–C	centrally coupled recursive feedback
RF–D	distributively coupled recursive feedback
RK–U	unified Runge-Kutta

NOMENCLATURE AND SYMBOLS

A	state gain matrix or attitude direction cosine matrix or panel area, depends on context
a	attitude control gain
aero	aerodynamic
B	input gain matrix
C_d	aerodynamic drag coefficient
cm	center of mass
cpi	center of pressure for the ith facet
d	drag
e	base of natural logarithm
$\vec{F}$	force
$f(X, t)$	mathematical function of state vector and time
G	gravitational constant
gg	gravity gradient
grav	gravity
H	height of spacecraft
I	moment of inertia matrix
i	integer variable
K	feedback gain matrix
k	loop gain
L	length of spacecraft
m	mass
n	integer variable
$\hat{n}$	unit vector normal to surface
R	orbit radius
$\vec{R}$	orbit radius vector
$\vec{r}_{cm}$	center of mass location vector
$\vec{r}_{cpi}$	center of pressure location vector for the ith facet
$\hat{r}$	unit vector in direction of orbit radius

NOMENCLATURE AND SYMBOLS (Continued)

sat	satellite
$\vec{T}$	torque
t	time
U	column matrix collection of all subsystem input vectors
u	subsystem input vector
V	orbit velocity
$\vec{V}$	orbit velocity vector
$\hat{v}$	unit vector in direction of orbit velocity
W	width of spacecraft
X	coordinate system axis or system-level state vector, depends on context
$\dot{X}$	time derivative of system-level state vector
x	coordinate system axis or subsystem state vector, depends on context
$\dot{x}$	time derivative of subsystem state vector
Y	coordinate system axis or system-level output vector, depends on context
y	coordinate system axis or subsystem output vector, depends on context
$\dot{Y}$	time derivative of system-level output vector
$\dot{y}$	time derivative of subsystem output vector
$\ddot{y}$	second time derivative of subsystem output vector
$y(0)$	initial value of $y(t)$
Z or z	axis of coordinate system
ρ	atmospheric density
ϕ	attitude angle
$\dot{\phi}$	attitude rate
ω	body rate
$\vec{\omega}$	angular velocity vector
$\dot{\vec{\omega}}$	angular velocity vector

TECHNICAL PUBLICATION

SYSTEM SIMULATION BY RECURSIVE FEEDBACK: COUPLING A SET OF STAND-ALONE SUBSYSTEM SIMULATIONS

1. INTRODUCTION

In design and development of space systems, digital dynamic simulations have potential to reduce physical testing requirements, increase confidence in systems that cannot be tested, and ensure that subsystem requirements are not more stringent than necessary. However, simulations that address interactions of multiple disciplines and subsystems can be difficult to build. Knowledge and expertise required at the physics level, especially for high-fidelity developments, do not normally reside within one person or organization for more than one discipline or subsystem. Simulations involving individual disciplines or subsystems usually exist or can be readily constructed, and system engineers often understand the qualitative nature of interactions among them; but a means of obtaining quantitative-coupled system-level results is often unavailable. Therefore, specific practical approaches to construction of high-fidelity system-level simulations are being developed from numerous perspectives across the engineering community.

Digital dynamic simulations are often built by deriving or constructing differential-algebraic equations that define the system model and implementing algorithms to solve those equations numerically, as a single coupled set, using one of several standard or modified numerical integration methods. In some cases, software is written in a language such as FORTRAN or C to define the equations, and an existing or slightly modified ordinary differential equation (ODE) solver, perhaps a Runge-Kutta implementation, is employed to perform the integration. In other cases, an in-house development such as Marshall System for Aerospace Simulation (MARSYAS) or a commercial off-the-shelf product such as MATLAB®/SIMULINK® is used to act as a higher level facilitator of what is ultimately a mathematically similar approach.

System-level digital solution of coupled "stand-alone" dynamical simulations is a fundamentally different approach to system simulation, and fundamentally different numerical procedures are required. Recursive feedback is a conceptual method from which a family of appropriate algorithms, processes, and specific numerical procedures can be derived.

Coupled-subsystem-simulation architecture provides near complete independence of stand-alone subsimulations and naturally facilitates high fidelity and broad scope through collaboration across interfaces that can be implemented in the same physical and engineering terms that define them in the real system. Such simplicity promotes clarity of communication and ease of understanding, both of which have many positive benefits. Individual subsimulations can potentially be implemented on separate, remote, and dissimilar computational platforms, and this portends numerous advantages and possibilities as computer network capabilities improve.

2. RECURSIVE FEEDBACK ALGORITHM

2.1 Background

Brief discussions of high-fidelity, multidiscipline simulation are set forth in references 1 and 2, which resulted from association with the Structures and Dynamics Laboratory High-Fidelity Multidiscipline Simulation Development team. Reference 1 attempts to illustrate the long-term goal of multidiscipline simulation by example, while reference 2 presents the goal in a more generalized way. Reference 3 documents development of an earlier FORTRAN general purpose dynamical simulation tool (ANASIM) that uses recursive feedback as the system solution algorithm. This tool allows one to construct a simulation by choosing system elements from a menu and specifying connections (signal flow) among them. The elements (integrators, summers, limiters, etc.) are, in effect, small stand-alone subsimulations. A definition and explanation of the method is presented, and an analytical demonstration of response computation for a first-order, single-state, constant-coefficient system with a step input is included. The demonstration, or mathematical "show that," is accomplished through symbolic algebraic application of the algorithm followed by direct comparison of results with the exact closed-form solution.

In reference 4, the method is analytically demonstrated for some simple examples within a variety of system classes. The first is the n-dimensional linear constant-coefficient case defined by the matrix vector equation $\dot{x}(t) = Ax(t) + Bu(t)$, which is a standard state-variable form. The remaining four are scalar (single state) cases that include two nonlinear, continuous cases; a linear time-varying coefficient case; and a numerically demonstrated case involving a state-dependent discontinuity and an algebraic loop. The analyses of reference 4 provide additional mathematical insights into the process, particularly with respect to convergence.

Analysis of the method is extended to coupling of stand-alone subsystems in references 5 and 6. Reference 5 mathematically defines and analytically demonstrates recursive feedback for generation of responses from a generic linear-dynamic system composed of internally inaccessible subsystems, and reference 6 continues beyond to include subsystems that have potential to produce algebraic loops at the system level because they contain direct feed-through of input.

References 5 and 6 specifically address process convergence and essentially guarantee that recursive feedback can be used as a system-level solution algorithm for simulation of linear systems composed of coupled stand-alone subsystems for a broad range of ordinary circumstances. Combined with the analyses of reference 4 and experience associated with ANASIM (ref. 3), it seems reasonable to expect that the method can be dependably applied to nonlinear system simulation. Only experience, and perhaps more analysis, will determine the effectiveness of extrapolation to these supposedly more stringent conditions.

2.2 Concept

Referring to figure 1, the subsystems are represented by independent, self-contained, time-domain, dynamical simulations that can be commanded to run from a given set of initial conditions over a predetermined segment of time when provided with input signals as functions of time over that interval. In general, the time segment, also referred to as a convergence interval, is short compared with the total desired system simulation duration, but may be significantly larger than the integration step size associated with a typical subsimulation. For each new stage of recursion, subsystem input signals are computed using original system-level input signals summed with system-level feedback (coupling) signals generated at the current stage. New (next recursion stage) subsystem responses are computed using new subsystem input signals in conjunction with the same (original) initial conditions. The process continues recursively until convergence is achieved. After convergence, initial conditions are replaced with final conditions and the process is restarted for the next segment of time. System response over the total desired duration of the simulation is the concatenated set of combined subsystem responses over many time segments.

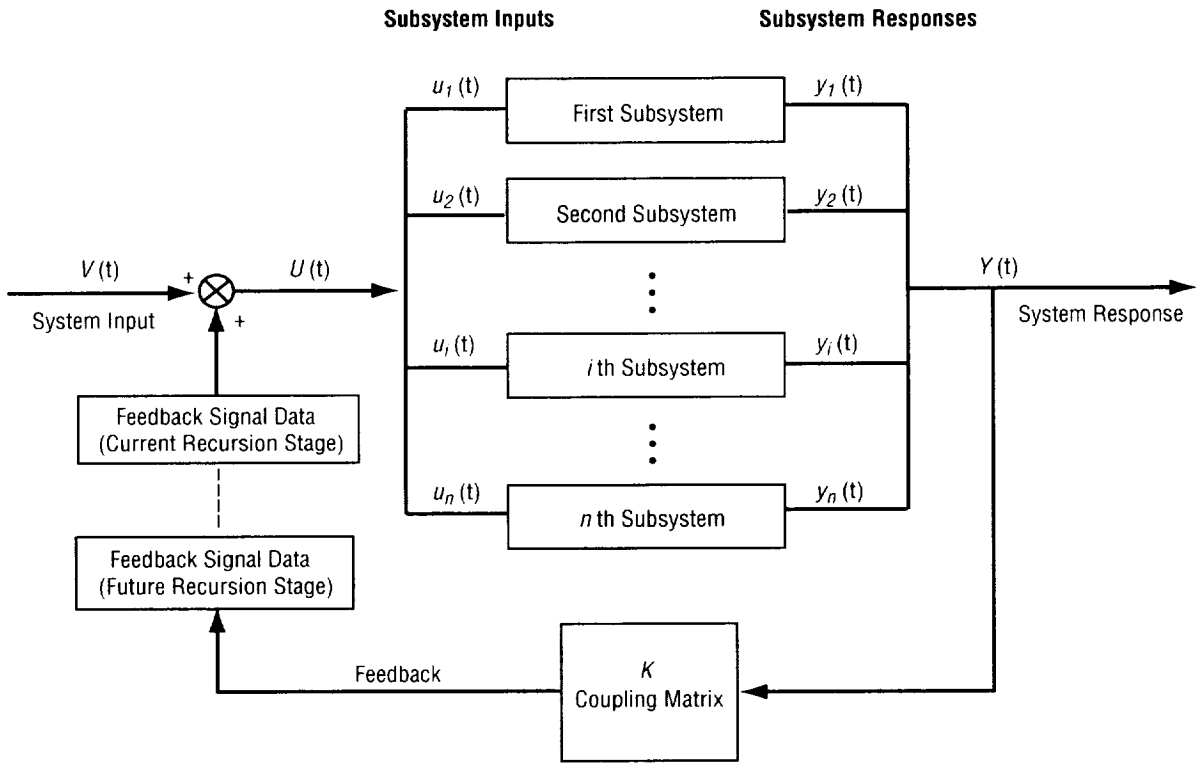


Figure 1. Schematic of recursive feedback process.

2.3 Insights for Specific Design

Because figure 1 describes a system of considerable generality whose complexity for a specific case can range from simple to great, further clarification of the process seems appropriate. The simplest "system" could arguably be a single integrator in the forward path, a constant multiplier (gain) in the backward (feedback) path, no (zero) system-level input, and a nonzero initial condition on the output. In that case, there is one linear subsystem, and that subsystem is a single integrator. The output function is the integral of the subsystem-level input function with respect to time, and can be generated independently and directly by simply computing the area under the input curve over the time interval of interest (convergence interval) and adding the initial condition. The coupling matrix becomes a scalar (value of the loop gain), and actually represents direct feedback rather than coupling. The process, however, does not explicitly make such a distinction, nor does it need to. The summing junction becomes moot since there is no system-level input. The recursive transfer point remains as depicted in figure 1, but does not exist in the analog system to be digitally simulated. The scalar differential equation that models the analog system is $\dot{y}(t) = ky(t)$, where t is time and k is the loop gain, and the exact closed-form solution for the output response is given by the exponential function

$$y(t) = y(0)\,[e^{kt}] \ , \tag{1}$$

where $y(0)$ represents the initial value of the output $y(t)$.

Applying recursive feedback to this system, the output is first computed as a function of time over the convergence interval as if there were no (zero) feedback. This function is referred to as the stage zero output response. Because the system input function is also zero in this case, the response is a constant function of time symbolically represented by

$$y^{(0)}(t) = y(0) \ , \tag{2}$$

where the parenthesized superscript is the stage index. Next, the stage zero feedback function is computed by multiplying the stage zero output response function by the loop gain k. After "moving" the feedback function data across the recursive transfer point and summing it with the system-level input function (zero), it becomes the stage one subsystem input function. Stage one response can now be computed by numerical integration (area under the curve) of the new (stage one) input function while applying the original initial condition $y(0)$. A symbolic representation of the result is

$$y^{(1)}(t) = y(0)[1 + kt] \ . \tag{3}$$

Repeating the process, symbolic representation of the stage two result is

$$y^{(2)}(t) = y(0)[1 + kt + \frac{(kt)^2}{2!}] \ . \tag{4}$$

By induction, the nth stage response is given by

$$y^{(n)}(t) = y(0)\left[1 + kt + \frac{(kt)^2}{2!} + \cdots + \frac{(kt)^n}{n!}\right] . \tag{5}$$

Clearly, if sufficiently continued, the recursive process yields a solution that approaches the exact solution $y(t) = y(0)[e^{kt}]$. In theory, the time interval (convergence interval) for this case can be as long as desired; in practice, it must be restricted because of numerical integration and roundoff errors. As mentioned earlier, system response over the problem duration is represented by a concatenation of responses over as many convergence intervals as needed, and a single convergence interval is measured in multiple integration steps. It is noted that each recursion stage improves accuracy of the approximate system response by adding the next higher order term in a series expansion of the exact system response. This type of behavior is seen in all of the linear system analyses of references 3–6. For the nonlinear systems of reference 4, each recursion stage adds higher order terms; but for a given number of correct terms in the series expansion, a greater number of recursion stages is required. For these systems, the convergence interval cannot be arbitrarily long, even without integration and roundoff errors, and the degree of restriction required depends on the initial condition.

Considering the system in figure 1 in its more general context, the off-diagonal elements of the coupling matrix represent the means by which output from one subsystem is processed and channeled to construct input for another. Because direct dependence of the input to any subsystem on the output of that same subsystem is likely to have been handled internally, the elements on the diagonal are likely to be zero; although, as demonstrated, the method does not require this to be the case. In development of a system simulation by coupling of subsystem simulations, the nature of the coupling matrix is, to a large degree, a design choice.

Consider, for example, the "system" to be an undamped oscillator governed by the scalar differential equation $\ddot{y}(t) + \omega^2 y(t) = 0$, where ω is the frequency of oscillation. A simulation of this system by recursive feedback can be set up in at least two very different ways. One design possibility is to define a single subsystem, best described as a double integrator, or two single integrators in series. The output function of the subsystem is determined by integrating its input function twice. The coupling matrix once again degenerates to a scalar constant, $-\omega^2$, and the system input function is zero. Assuming a nonzero initial condition $y(0)$ on the output (displacement), application of recursive feedback to this system results in a truncated series definition of $y(t)$, and at the nth recursion stage, a symbolic representation is

$$y^{(n)}(t) = y(0)\left[1 - \omega^2\frac{t^2}{2!} + \omega^4\frac{t^4}{4!} + \cdots + (-1)^n\omega^{2n}\frac{t^{2n}}{(2n)!}\right] . \tag{6}$$

Then for a sufficiently large number of recursion stages, the approximate solution approaches the exact solution $y(t) = y(0)[\cos\omega t]$. Once again the number of terms in the series corresponds directly to the number of recursion stages, but the order of the series with respect to time is twice the number of recursion stages.

In contrast, a second design possibility is to define two subsystems that are both single integrators. The governing differential equation is now given by the two-dimensional matrix-vector equation

$$\dot{Y}(t) = KY(t) \ , \tag{7}$$

the coupling matrix is given by

$$K = \begin{bmatrix} 0 & 1 \\ -\omega^2 & 0 \end{bmatrix} , \tag{8}$$

and the output function is given by

$$Y(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} , \tag{9}$$

where $y_1(t)$ represents displacement and $y_2(t)$ represents velocity. Application of recursive feedback to this system results in a truncated series definition of $Y(t)$. At the nth recursion stage, the numerically computed approximate system response function is symbolically represented by

$$Y^{(n)}(t) = \left[I + Kt + K^2 \frac{t^2}{2!} + \cdots + K^n \frac{t^n}{n!} \right] Y(0) \ . \tag{10}$$

Again, each numerical recursion stage, in effect, adds one term to the truncated series approximation of the exact solution.

For the two-subsystem, twin-integrator approach, it takes two recursion stages to reach the same order of approximation in time that was reached in one stage with the single-subsystem, double-integrator approach. However, the twin integrators represent independent operations that can be performed in parallel, while the double integrator represents two operations that must be performed in series. In the double integrator case, velocity, $\dot{y}(t)$, does not appear at the system level, only one function crosses the recursive transfer point, and only one function is checked for convergence. In the twin integrator case, velocity, $y_2(t)$, appears at the system level, both $y_1(t)$ and $y_2(t)$ cross the recursive transfer point, and both functions can be checked for convergence at the system level. Clearly, in a more complex situation, the relative merits of these approaches become simulation design and performance issues.

For the broader range of systems contained within the framework of figure 1, the nature of the coupling matrix is a function of many possible simulation design choices. In its most basic form, a coupler would simply route signals from one subsystem to another without modifying them. In its most complex form, it could become a multidimensioned nonlinear operator as well. Typical coupler functions are likely to be coordinate transformations, gain multiplications, interpolations, and other data modification or routing tasks that must be performed to make proper interface connections among a set of predefined or existing subsimulations.

As implied by the double integrator example (when viewed as two single integrators in series), it is also possible to distribute the coupling functions throughout the system. In a particular situation, one might choose to view several subsimulations as a combined set if the nature of the coupling among them is such that an output response can be determined from an input function without necessity for recursion (there is no feedback loop internal to the set). In that case, processing information as it is transferred from one set member to another represents a coupling function that does not reside as an element of a centralized coupling matrix. Utilizing the powers of block diagram algebra or signal flow analysis, many arrangements of subsystems and couplers may be possible for representation of a given system. Such design freedom can have a significant impact on the configuration, performance, and efficiency of the simulation.

To be of practical value in dynamical systems simulation, the recursive feedback concept must be applicable to nonlinear models. The "guarantees" and insights that come with analyses like that of reference 6 are weakened at best and totally lost at worst without the linear model assumption. However, one does have a guarantee of solution existence and uniqueness for a broad range of sufficiently well-behaved nonlinear systems (ref. 7). A converged recursive feedback process is indicative of a solution with exception of questions concerning discretization errors, roundoff errors, inappropriate tolerances, etc. The potential for convergence can normally be improved by reducing the length of the convergence interval, not unlike reducing integration stepsize in a conventional situation when satisfactory results are not achieved.

3. EXAMPLE SYSTEM

The example system is designed to be simple but to reflect real possibilities. It consists of a satellite that has an attitude control system, is in low-Earth orbit, and is experiencing aerodynamic effects in addition to gravitational effects. The spacecraft is idealized as a rigid body of rectangular "box" shape, illustrated in figure 2. The center of mass is offset from the geometric center by a small amount in all three axes, and moments of inertia are such that there are two large but somewhat unequal principal values, while the third principal value is much smaller. The principal axes are normal to the box surfaces so that all cross products of inertia are zero. The models are based on constant density atmosphere, spherical Earth (inverse square) gravity, and linear attitude control for small motion about an inertial reference. Attitude control can be partially or totally removed so that the body is allowed to tumble under the influence of gravity and aerodynamics. Gravitational, aerodynamic, and control torques affect the rotational dynamics of the rigid body, while gravitational and aerodynamic forces affect the translational dynamics. Both translational and rotational dynamics affect aerodynamic forces and torques, so a feedback loop is apparent. This example falls in the general category of nonlinear, continuous systems.

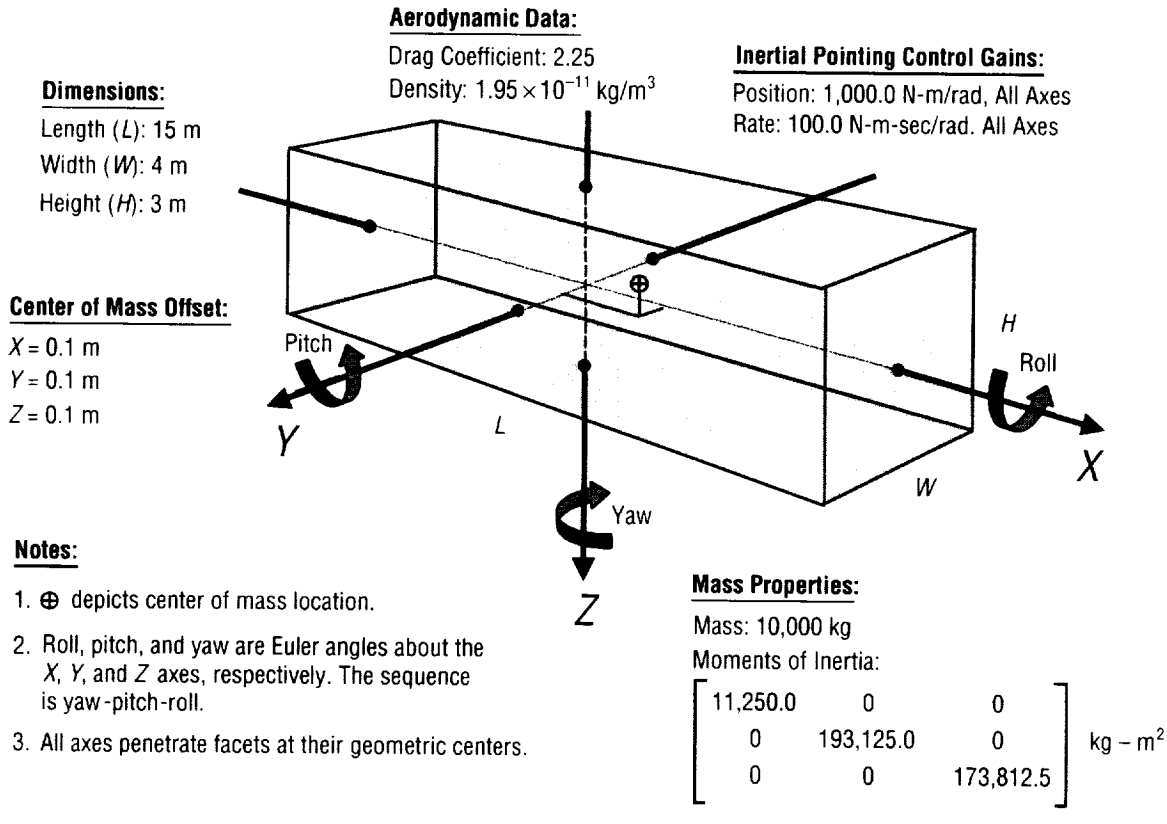


Figure 2. Spacecraft properties and geometric body-fixed coordinate system.

4. MATHEMATICAL MODELS

4.1 Orbit Dynamics

The orbit dynamics is modeled by

$$m_{\text{sat}} \frac{d^2\vec{\text{R}}}{dt^2} = \vec{\text{F}}_{\text{grav}} + \vec{\text{F}}_{\text{aero}} \quad , \tag{11}$$

where

$$\vec{\text{F}}_{\text{grav}} = -\frac{Gm_{\text{Earth}}m_{\text{sat}}}{R^3}\vec{\text{R}} \quad . \tag{12}$$

G is the universal gravitational constant, m_{Earth} and m_{sat} are masses of the Earth and the satellite. $\vec{\text{R}}$, the orbit radius vector, extends from the center of the Earth to the center of mass of the satellite and is expressed in an Earth-centered reference frame that is defined to be nonrotating and assumed to be nonaccelerating. $\vec{\text{F}}_{\text{aero}}$, the aerodynamic force, is derived from an aerodynamics model to be discussed later. This development, as well as the attitude dynamics development, is standard and of the general type found in many dynamics texts such as reference 8.

4.2 Attitude Dynamics

The rotational dynamics is modeled as a rigid body by

$$I\dot{\vec{\omega}} + \vec{\omega} \times I\vec{\omega} = \vec{\text{T}}_{\text{control}} + \vec{\text{T}}_{\text{gg}} + \vec{\text{T}}_{\text{aero}} \quad , \tag{13}$$

where I is the moment of inertia matrix, $\vec{\omega}$ is the angular velocity vector, $\vec{\text{T}}_{\text{gg}}$, is gravity gradient torque, $\vec{\text{T}}_{\text{aero}}$ is aerodynamic torque, and all quantities are expressed in a body-fixed coordinate system whose origin is at the center of mass.

The attitude is defined by a set of Euler angles $\left[\phi_x, \phi_y, \phi_z\right]$ that indicate angles traversed by a hypothetical reference frame as it is rotated from coincidence with an inertial (nonrotating) attitude reference frame to coincidence with the body frame using three single-axis rotations in a z-y-x sequence. The instantaneous Euler angle rates are then related to body rates by

$$\begin{bmatrix} \dot{\phi}_x \\ \dot{\phi}_y \\ \dot{\phi}_z \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi_x \tan\phi_y & \cos\phi_x \tan\phi_y \\ 0 & \cos\phi_x & -\sin\phi_x \\ 0 & \sin\phi_x \sec\phi_y & \cos\phi_x \sec\phi_y \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \quad . \tag{14}$$

Although this particular formulation has a singularity at $\phi_y = \pm\pi/2$, it is valid except at that point, and can serve as the basis for a model if an adequate neighborhood of that point is avoided.

A vector expressed in inertial coordinates is transformed to body coordinates through multiplication of the corresponding column of inertial components by the direction cosine matrix

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi_x & \sin\phi_x \\ 0 & -\sin\phi_x & \cos\phi_x \end{bmatrix} \begin{bmatrix} \cos\phi_y & 0 & -\sin\phi_y \\ 0 & 1 & 0 \\ \sin\phi_y & 0 & \cos\phi_y \end{bmatrix} \begin{bmatrix} \cos\phi_z & \sin\phi_z & 0 \\ -\sin\phi_z & \cos\phi_z & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{15}$$

The attitude control system is modeled by

$$-\bar{T}_{\text{control}} \Leftarrow \begin{bmatrix} a_{0x}\phi_x + a_{1x}\omega_x \\ a_{0y}\phi_y + a_{1y}\omega_y \\ a_{0z}\phi_z + a_{1z}\omega_z \end{bmatrix}, \tag{16}$$

where ϕ, ω, and a are associated with attitude angles, body rates, and control gains, respectively, and the subscripts are self-explanatory.

The gravity gradient torque is modeled by[9]

$$\bar{T}_{gg} = \frac{Gm_{\text{Earth}}}{R^3}\hat{r} \times I \cdot \hat{r}, \tag{17}$$

where $\hat{r}$ is a unit vector corresponding to $\bar{R}$ expressed in body coordinates.

4.3 Aerodynamics

A box with six rectangular side panels (facets) defines the satellite shape (fig. 2). Aerodynamic drag forces associated with the ith panel are approximated empirically by the formula[9]

$$\bar{F}_i = \begin{cases} -\frac{1}{2}\rho V^2 C_d A_i (\hat{n}_i \cdot \hat{v})\hat{v} & \text{if} \quad \hat{n} \cdot \hat{v} \geq 0 \\ 0 & \text{if} \quad \hat{n} \cdot \hat{v} \leq 0 \end{cases}, \tag{18}$$

where ρ is atmospheric density, V is velocity magnitude relative to the atmosphere, C_d is a drag coefficient, A is panel area, $\hat{n}$ is a unit vector normal to the panel and directed outward from the box, and $\hat{v}$ is a unit vector corresponding to $\vec{V}$. If rotation of the Earth is neglected, no distinction between relative and absolute velocity need be made. The total aerodynamic force is given by

$$\vec{F} = \sum_{i=1}^{6} \vec{F}_i \quad . \tag{19}$$

While the vector relationships of equations (18) and (19) are independent of any particular coordinate system, the most convenient choice is a body-fixed "geometric" coordinate system (fig. 2) whose origin and axes are defined from geometry (shape) considerations alone. The aerodynamic force as used in equation (11) is most appropriately expressed in inertial coordinates. Appropriate coordinate transformation depends on the matrix A, defined by equation (15), and on the orientation of the body frame with respect to the geometric frame. As illustrated in figure 3, the axes of the body frame are not rotated with respect to the geometric frame, so the relevant transformation is identity.

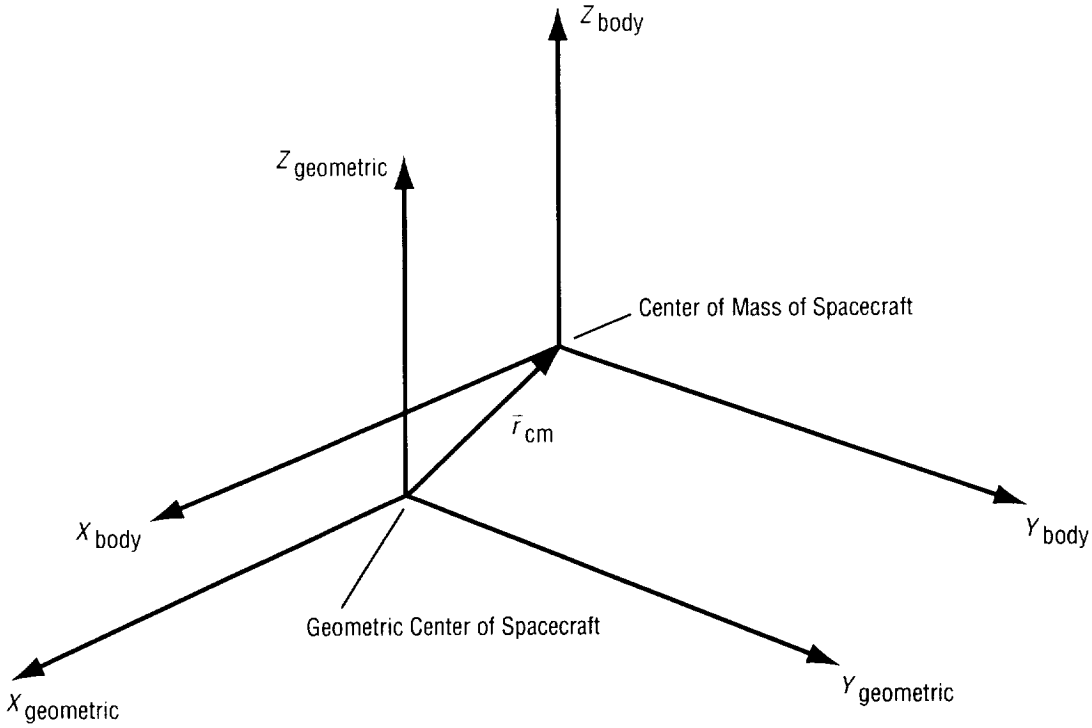


Figure 3. Relationship of body frame to geometric frame.

The total aerodynamic torque about the origin of the geometric frame is calculated from six contributing components and is given by

$$\bar{T} = \sum_{i=1}^{6} \bar{r}_{cpi} \times \bar{F}_i \quad , \tag{20}$$

where $\bar{r}_{cpi}$ is a vector from the origin of the geometric coordinate system to the center of pressure for the ith facet. The total aerodynamic torque about the center of mass of the spacecraft, as used in equation (13), is given by

$$\bar{T}_{cm} = \vec{T} - (\bar{r}_{cm} \times \bar{F}) \quad . \tag{21}$$

where $\bar{r}_{cm}$ is a vector from the origin of the geometric coordinate system to the center of mass (fig. 3).

5. SUBSYSTEM SIMULATIONS

Simulations for each of the three subsystems (orbit dynamics, attitude dynamics, and aerodynamics) have been constructed as functionally separate entities, although they are implemented as parts of one single FORTRAN computer program. The attitude and orbit dynamics simulations both have 3 degrees of freedom, and each has its own separate implementation of a Runge-Kutta numerical integration process. In both cases, the three second-order differential equations have been rearranged to six first-order equations for interface with the integration algorithm. The aerodynamics simulation has no integration process because the model is purely logical-algebraic. All three simulations are "stand-alone" because each, in principle, is capable of producing output signals (functions of time) from input signals and initial conditions. Initial conditions, of course, do not apply to the logical-algebraic aerodynamic simulation. Input and output signals are represented by a sequence of linearly connected points equally spaced in time. Additional points are obtained through interpolation by each simulation if needed.

The orbit dynamics simulation input signal is aerodynamic force, and output signals are orbit radius and velocity vectors. The model, defined by equations (11) and (12), considers a point mass, inverse square gravity, and an external force (aerodynamics) that is an arbitrary but known function of time. All orbit dynamics simulation variables are referenced to an Earth-centered inertial coordinate system.

The attitude dynamics model input signals are the orbit radius vector expressed in inertial coordinates, and aerodynamic torque expressed in body coordinates. Output signals are body rates and attitude angles. The model, defined by equations (13)–(17), considers rotational motion of a rigid body, gravity gradient torque, attitude control system torque, and an external torque (aerodynamics) that is an arbitrary but known function of time.

The aerodynamics simulation input signals are orbit radius vector in inertial coordinates and orbit velocity vector in body coordinates. Output signals are aerodynamic force and torque expressed in geometric coordinates. The model is logical-algebraic and is defined by equations (18)–(21).

6. SYSTEM SIMULATIONS

Three system simulations have been constructed. The first simulation uses a centralized coupling implementation of recursive feedback, while the second simulation uses a distributed coupling implementation. The third simulation uses a conventional first-order, single-equation set approach and is the standard against which the recursive feedback approaches are compared.

By design, the function of a coupler is to perform coordinate transformations or other types of data conversions needed to make consistent, correct connections among the subsystem simulations. A single coupler can perform all coupling functions from one central location, or a set of couplers that perform less extensive functions can be distributed throughout the system. Coupling signals, like input and output signals, are represented by a sequence of points equally spaced in time.

Figures 4 and 5 illustrate the centrally coupled system simulation. The subsimulations represent independent processes that can, in principle, be executed in parallel over the timespan of each convergence interval. Because the primary purpose is to investigate functionality rather than efficiency, no attempt has been made to use more than one processor in the single-computer FORTRAN implementation. Therefore, the parallel computation advantage is lost, and quantitative comparison of efficiency with the other simulations is not possible.

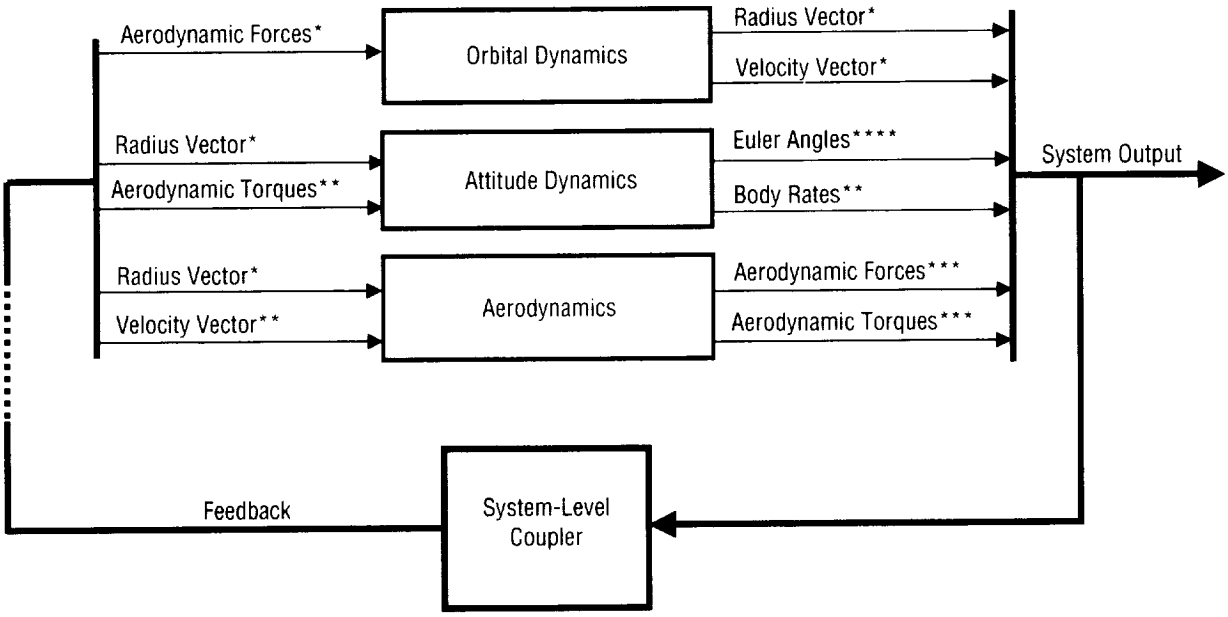
Figure 6 illustrates the distributively coupled system simulation. Before applying the idea of recursive feedback, the subsystems have been arranged so that, to the extent possible, feedback is minimized. Because there is an information loop that ultimately must be closed, it is not possible to completely eliminate feedback. The subsimulations are identical to those of the centrally coupled system, but the system-level numerical process is substantially affected because, among other things, the number of scalar signal paths that cross the recursive transfer point has changed. This simulation was developed in part because the centrally coupled simulation, developed first, is noticeably slower than the standard method benchmark. More importantly, it serves to confirm functionality of the different coupling architectures.

The conventional simulation is implemented from the combined set of equations that define the subsystem models. They have been collected and rearranged, by hand, to a single set of 12 first-order differential equations, an arrangement sometimes called a "state variables" approach to system formulation. In that context, the state vector is defined as

$$X = \left[R \mid V \mid \phi \mid \omega \right]^T \quad , \tag{22}$$

where R, V, ϕ, and ω are three-element row vectors representing orbit radius, velocity, Euler angles, and body rates. The system differential equations can now be represented as a first-order set by

$$\dot{X} = f(X, t) \quad , \tag{23}$$



..... Recursive Transfer Point

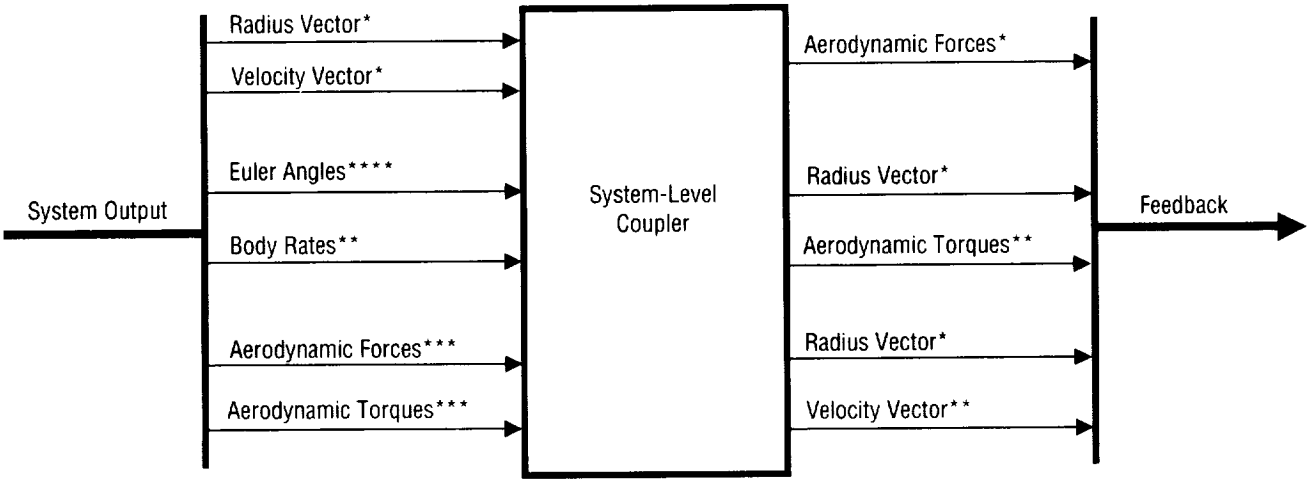
*Inertial Coordinates

**Body Coordinates (cm Origin)

***Geometric Coordinates

****From Inertial to Body Frame, 3–2–1 Rotation Sequence

Figure 4. Centrally coupled simulation diagram.



*Inertial Coordinates

**Body Coordinates (cm Origin)

***Geometric Coordinates

****From Inertial to Body Frame, 3–2–1 Rotation Sequence

Figure 5. Function of system-level coupler.

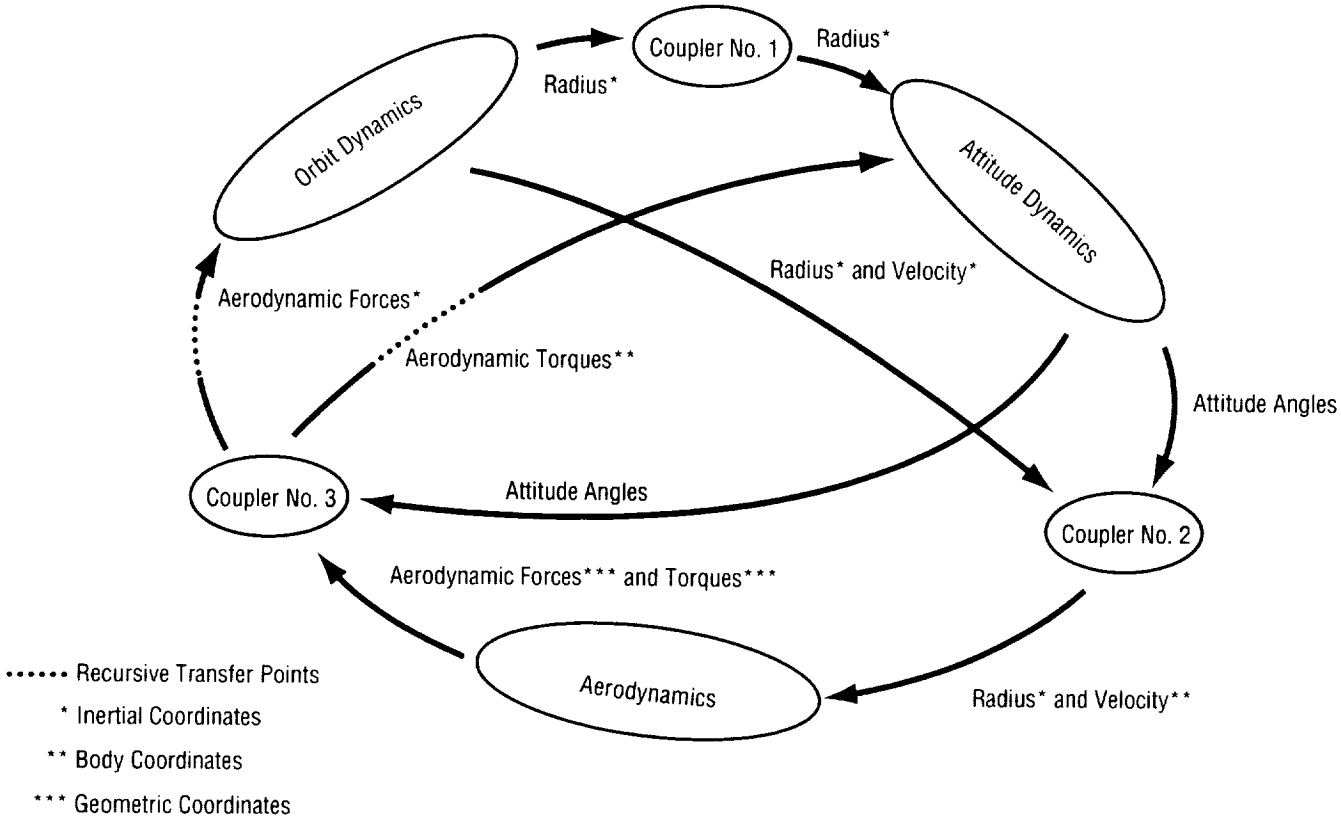


Figure 6. Distributively coupled simulation diagram.

where time has been explicitly included only for completeness of the form. In the current simulation, time is only implicitly involved, because there are no system forces or torques that are explicit functions of time. A fourth-order Runge-Kutta numerical integration algorithm specifically designed to solve systems in the form of equation (13) is employed to propagate the system responses.

All system and subsystem simulations were specially developed for this project. They are written in FORTRAN, all data were generated on the TD55/eds2 UNIX platform at Marshall Space Flight Center (MSFC). Plots were originally produced with an in-house program (developed by Warren Adams, TD55, MSFC) on the same machine. Plot images were migrated to a Windows desktop PC for publication. A full complement of plots is included in appendix A, and listings of the simulation programs are included in appendix B.

7. NUMERICAL RESULTS

7.1 Run Case Definitions

All data cases simulate motion of the same spacecraft starting from the same orbit and attitude related initial conditions. The orbit is initialized so that in the absence of aerodynamic drag it would be circular at an altitude of 300 km and an inclination of 30 deg. The initialization point is in the X–Z plane of the coordinate system shown in figure 7. Spacecraft mass properties, dimensions, and control gains are given in figure 2. Initial attitude angles and body rates are zero. Other data constants and physical constants are identified in the computer listings. Case variations are developed from three attitude control system configurations and three simulation methods resulting in a nine-element run matrix. The attitude control configurations are listed as follows:

(1) Full inertial control
(2) Partial inertial (two-axis) control
(3) No control (tumble).

Case numbers correspond directly to the control system configuration list entry numbers, and method variations are associated with each case. The simulation methods are:

(1) Centrally coupled recursive feedback (RF–C)
(2) Distributively coupled recursive feedback (RF–D)
(3) Conventional (unified) Runge-Kutta (RK–U).

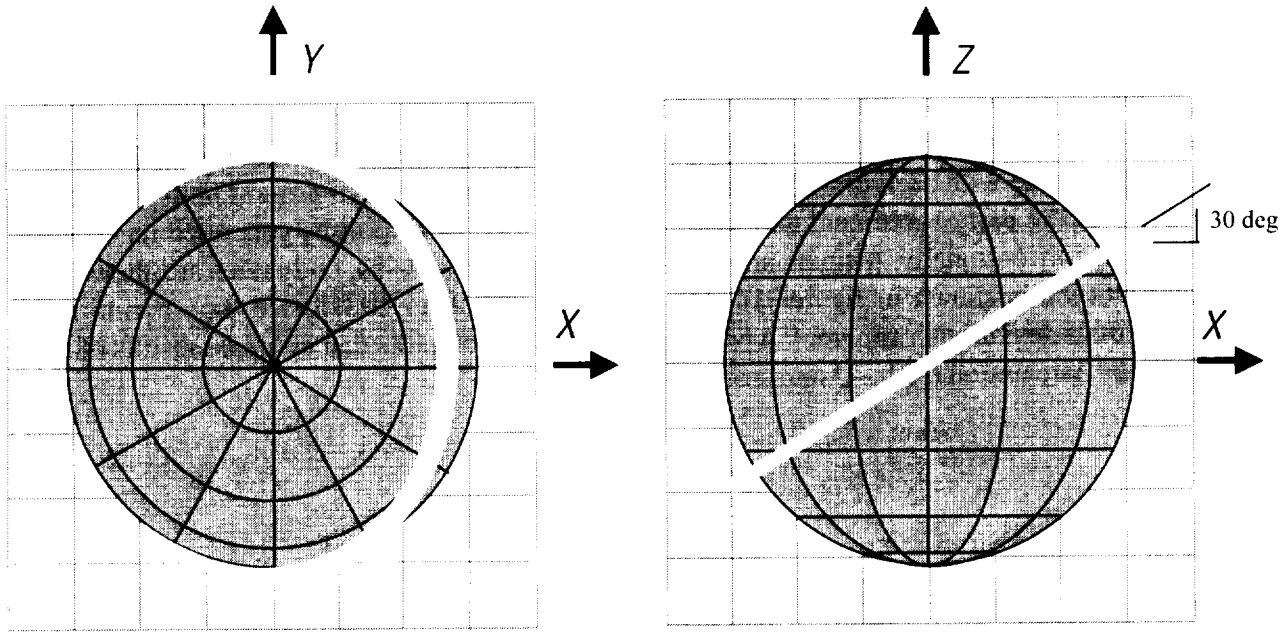


Figure 7. Inertial coordinate system/initial orbit plane geometry.

Numerical algorithm parameter settings were the same for all cases. Both recursive methods used five sample points per convergence interval (including both end points), and the interval length was 0.4 sec. Relative (normalized) and absolute convergence tolerances were both 10^{-12} for the RF–C and 10^{-14} for the RF–D. The orbit and attitude subsimulations both used a fourth-order Runge-Kutta algorithm with a step size of 0.1 sec, while the aerodynamics subsimulation computed aerodynamic effects at a 0.1-sec sample interval. The conventional simulation used the same Runge-Kutta algorithm with a step size of 0.1 sec.

Results for all possible (nine) combinations of control system configuration and solution methods were generated. For case 1, in the absence of any well-defined requirement, a set of control system gains was chosen that produce reasonable (small angle) attitude dynamics responses. For case 2, the Z-axis control gains were set to zero; and for case 3, all control gains were set to zero. Simulation responses have been generated for 60,000 sec of problem time using all solution approaches for all cases.

7.2 Response Comparisons

The responses for case 1 (full control) are identical over the full 60,000-sec duration of the simulation runs for all solution approaches (RF–C, RF–D, and RK–U). Appendix A.1 contains 108 plots that compare RF–D results to conventional RK–U results for case 1. Radius, velocity, attitude angles, body rates, aerodynamic forces, aerodynamic torques, and altitude are shown as functions of time. The general format of the appendix presentation is to show each response over time intervals of 0–20,000, 20,000–40,000, and 40,000–60,000 sec, and additionally, to show some selected plots that cover smaller time intervals of particular interest.

The responses for case 2 (two-axis control) are, like case 1, identical over the full 60,000-sec duration of the simulation runs for all solution approaches. Appendix A.2 contains 92 plots that compare RF–D results to conventional RK–U results for case 2. As in the appendix presentation for case 1, radius, velocity, attitude angles, body rates, aerodynamic forces, aerodynamic torques, and altitude are shown as functions of time, and the presentation format is also similar to that used for case 1.

For case 3 (tumble), the RF–C, RF–D, and RK–U simulation responses all agree for approximately 45,000 sec of the duration; after that, the RF–C and RF–D responses begin to diverge from the RK–U while they continue to agree with each other for the full 60,000 sec. Figures 8–16 present plots selected specifically to illustrate the early agreement and subsequent deviation of responses generated recursively from those generated conventionally. Attitude angles are shown in figures 8–10, body rates are shown in figures 11–13, and altitude is shown in figures 14–16. Only the 40,000- to 60,000-sec time period is covered because no earlier deviation is apparent. Appendix A.3 contains 60 plots that compare results of the RF–D approach to the conventional RK–U approach for this case.

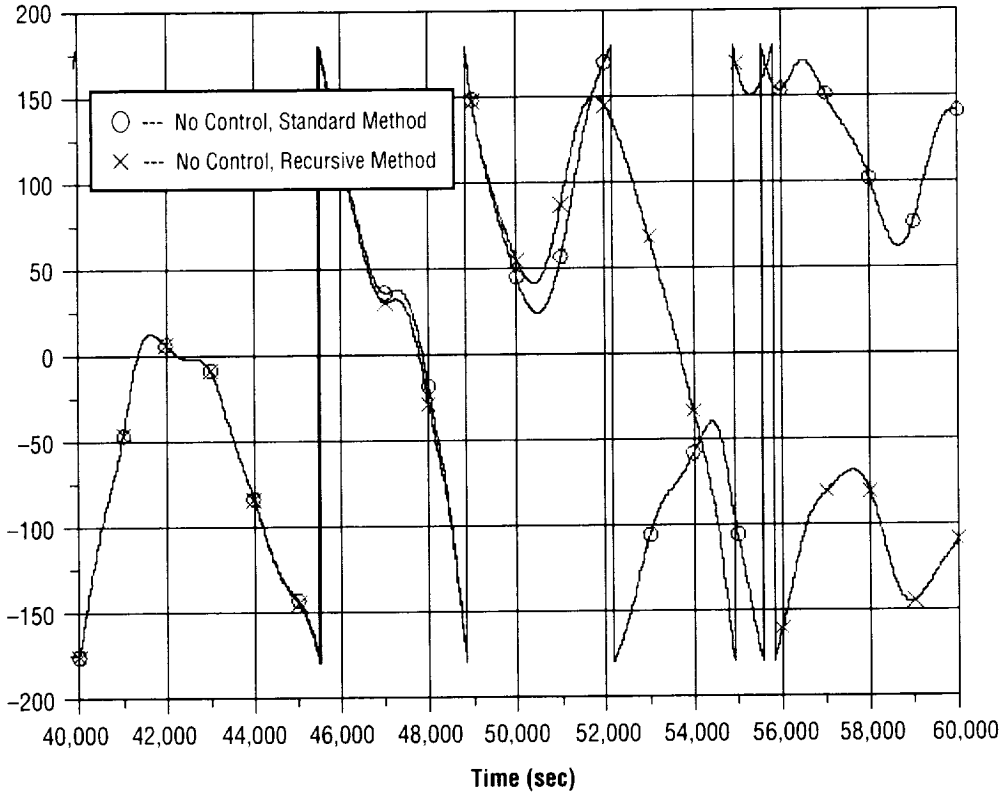


Figure 8. Case 3, Euler angle, *X* axis (deg).

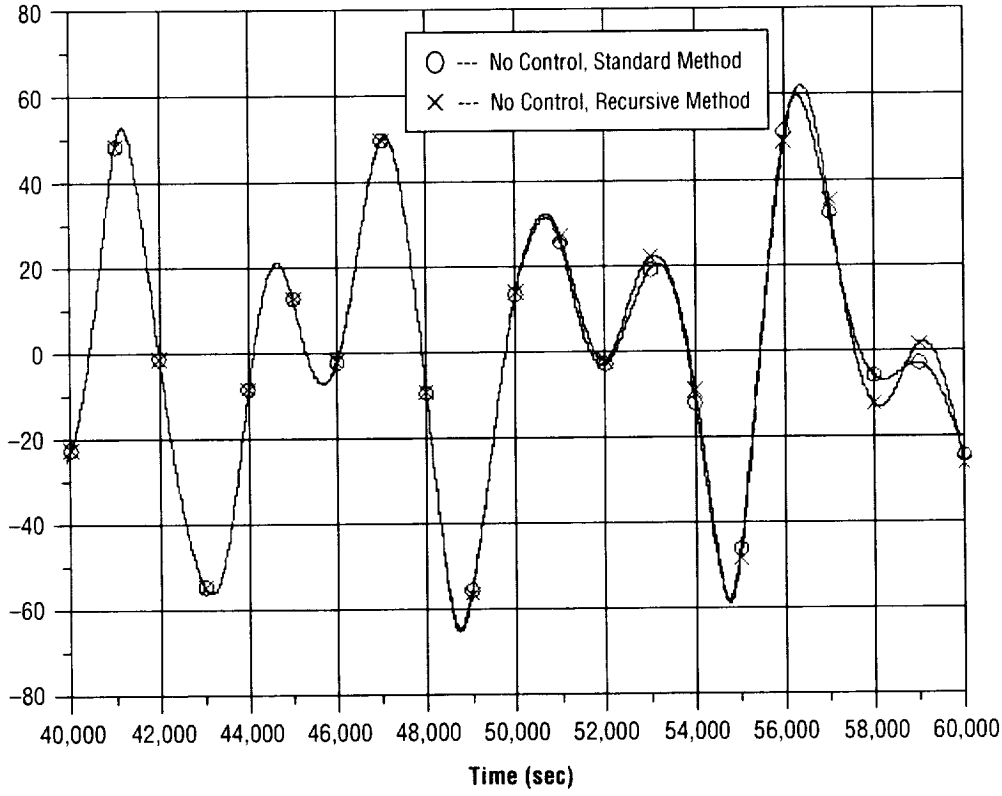


Figure 9. Case 3, Euler angle, *Y* axis (deg).

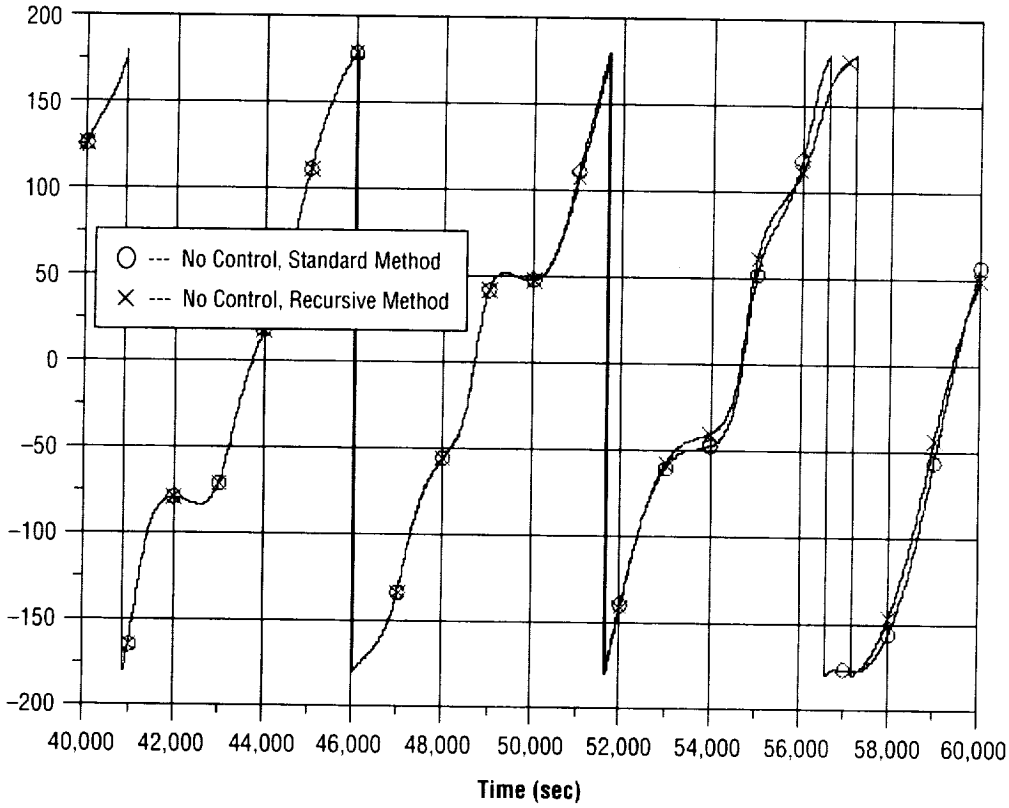


Figure 10. Case 3, Euler angle, Z axis (deg).

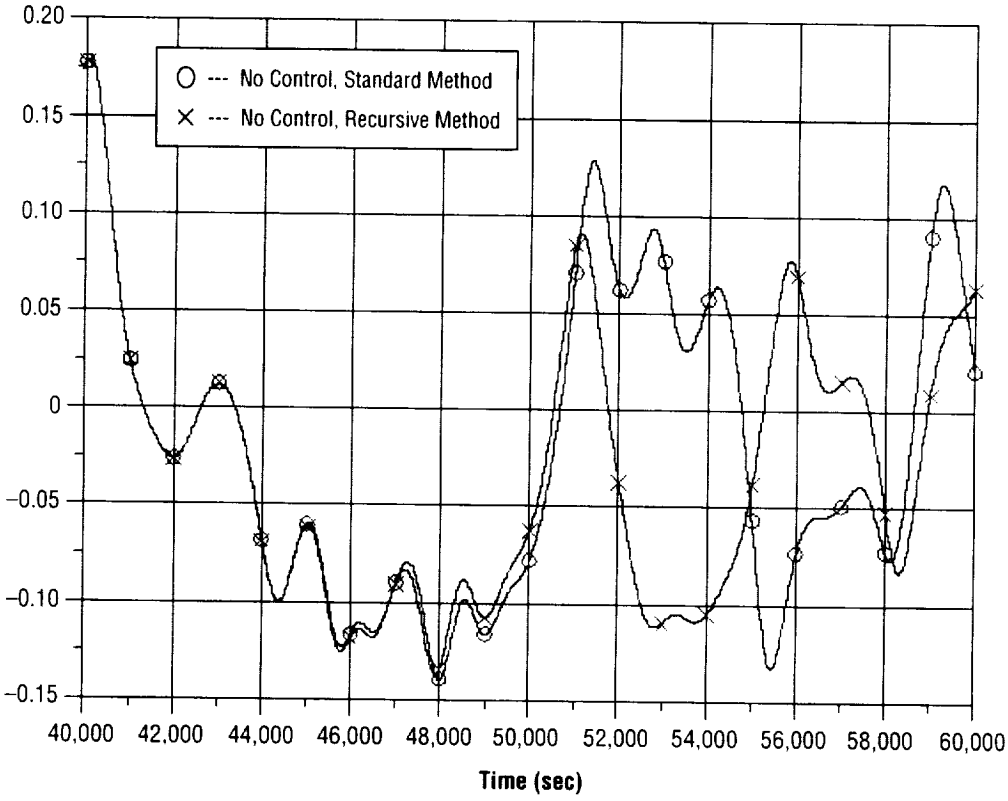


Figure 11. Case 3, body rate, X axis (deg).

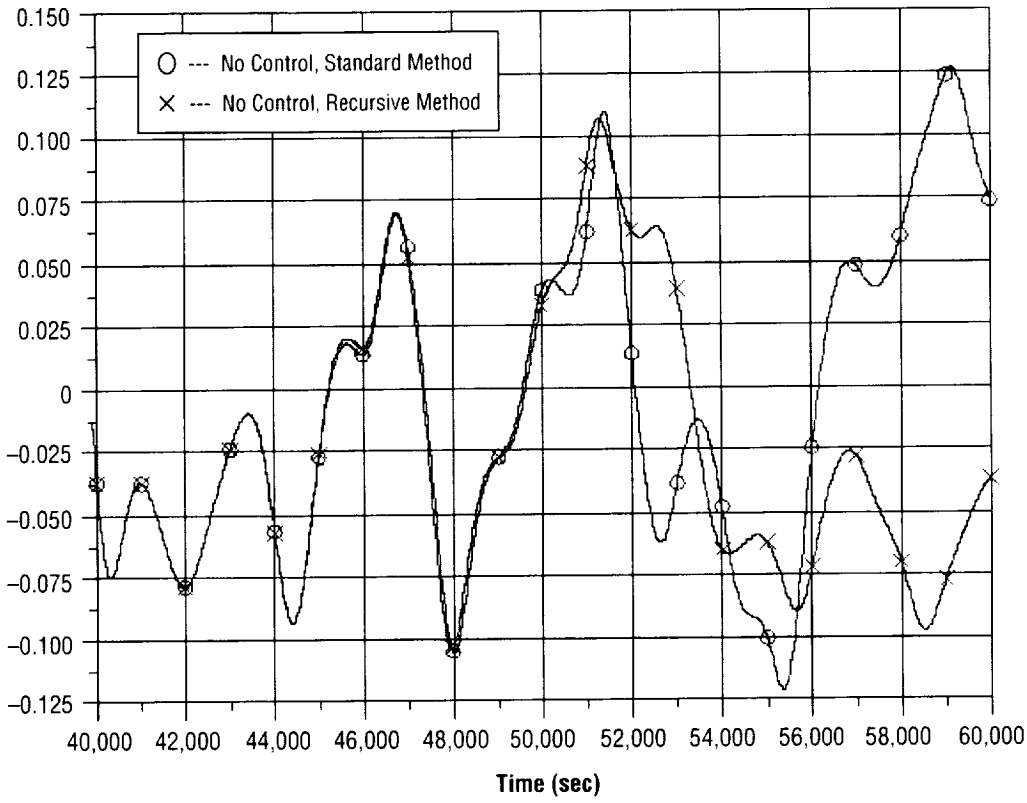


Figure 12. Case 3, body rate, Y axis (deg).

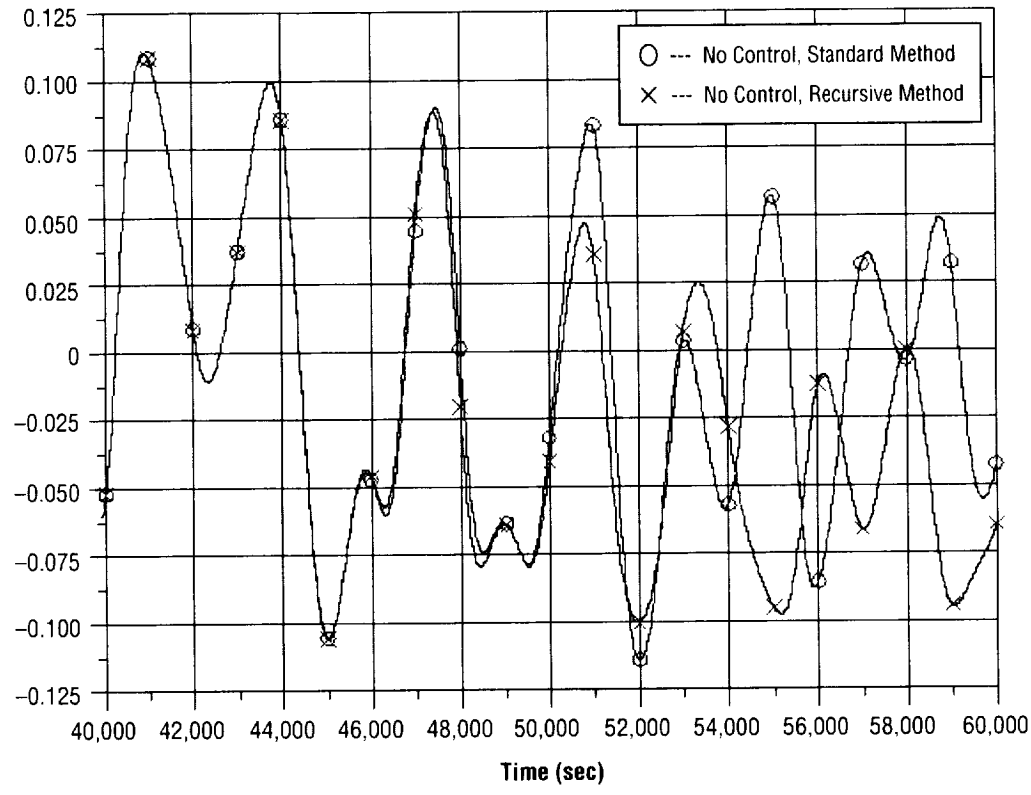


Figure 13. Case 3, body rate, Z axis (deg).

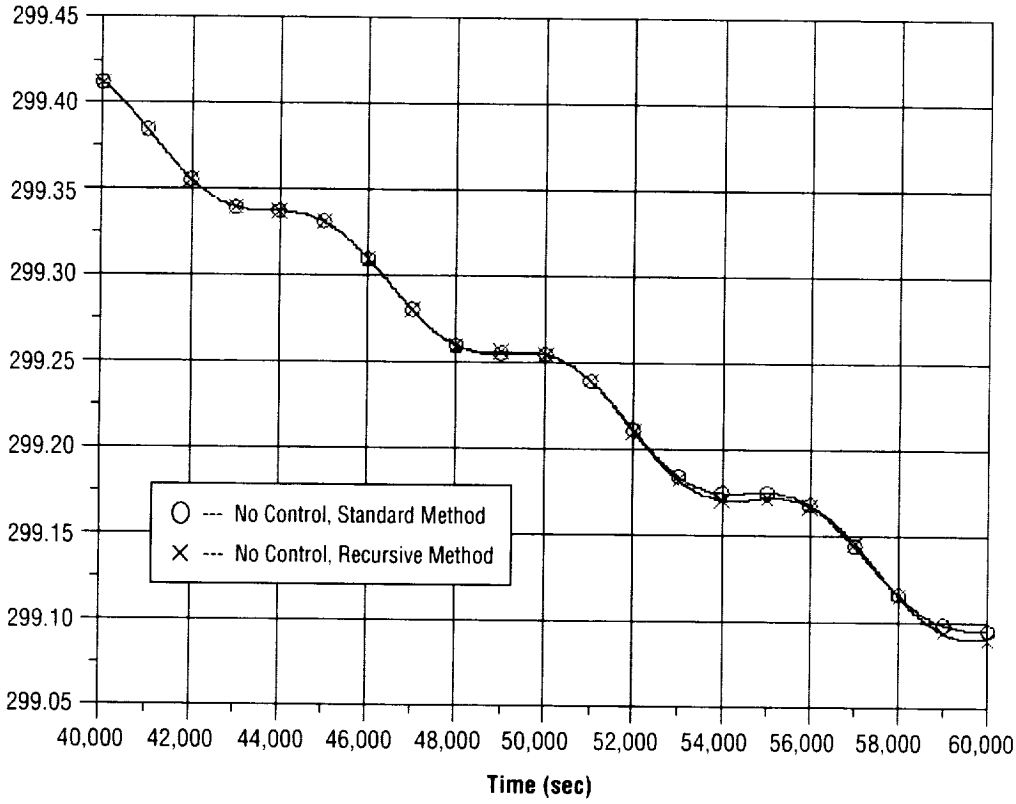


Figure 14. Case 3, altitude, first time segment (km).

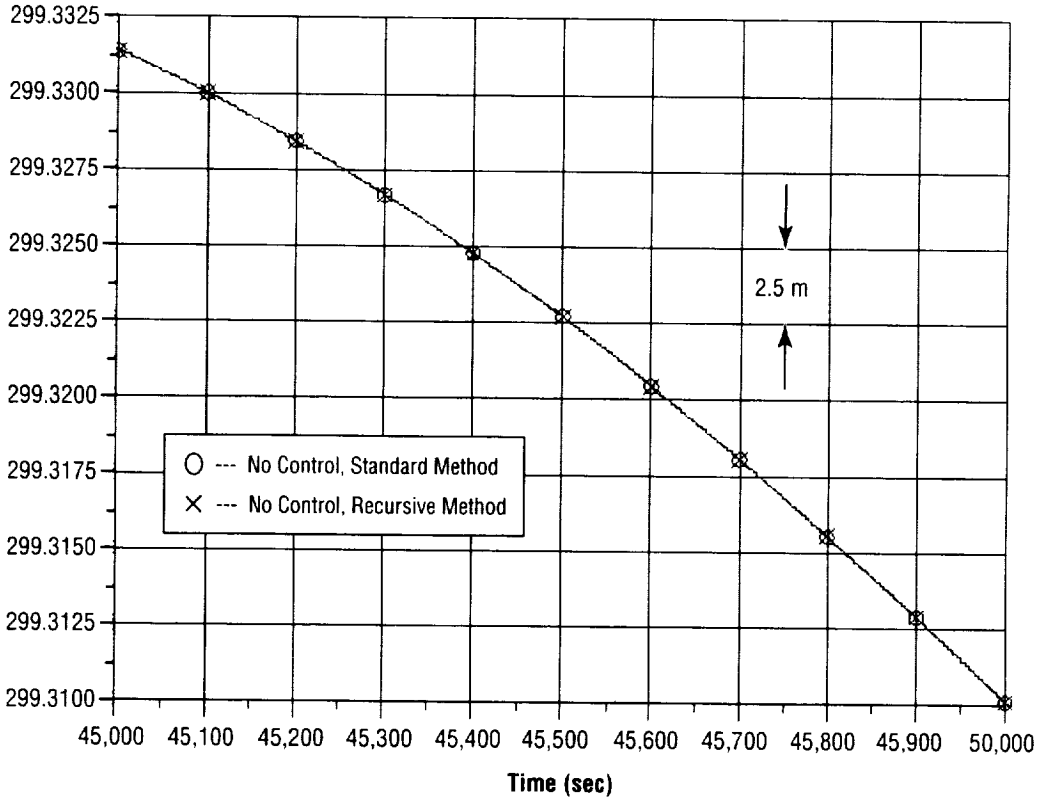


Figure 15. Case 3, altitude, second time segment (km).

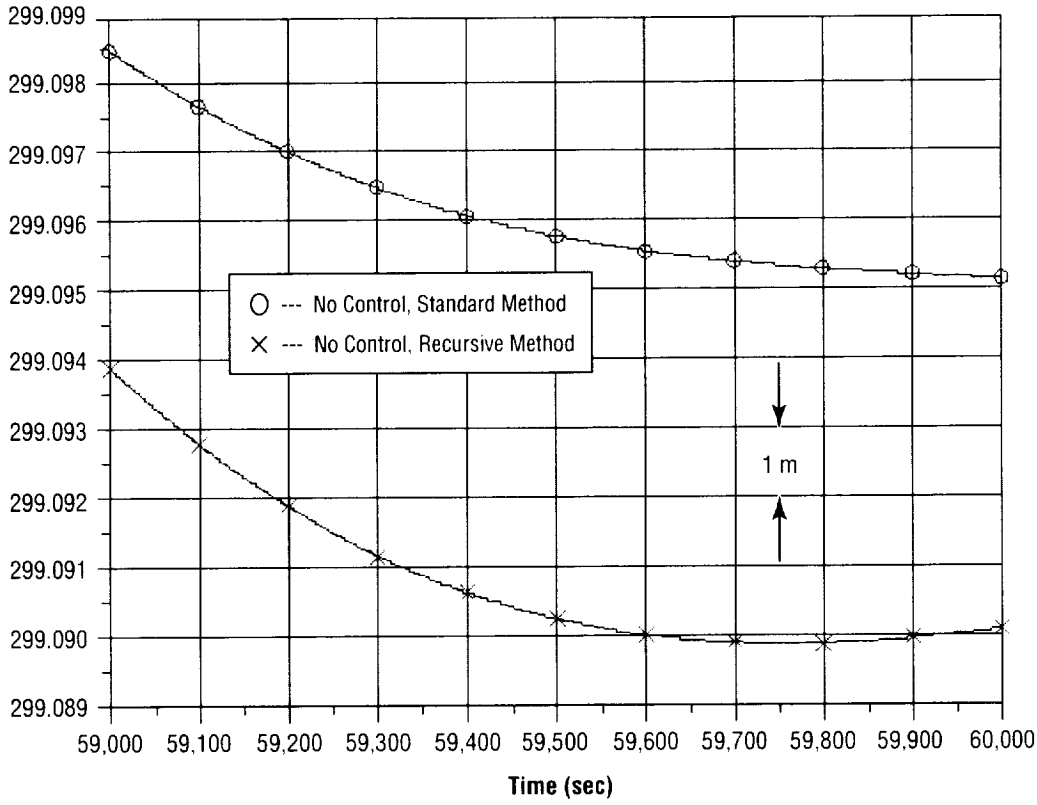


Figure 16. Case 3, altitude, third time segment (km).

Altitude results for all control system configurations and two solution methods (RF–D and RK–U) are superposed in figures 17–19. The main point of illustration is the variation in orbit decay rate as a function of attitude control. Figure 17 shows the full 60,000-sec run duration, while figures 18 and 19 show shorter time segments taken near the end of the run. Figures 17–19 illustrate again the agreement of recursive and conventional solution methods for cases 1 and 2, while showing the method-induced deviation apparent in case 3, the no control (tumble) case.

Appendix A.4 contains 42 plots that compare results from the two recursive approaches (RF–C and RF–D) to each other for all three cases. These plots are presented primarily for completeness since, for all practical purposes, the responses are identical for both methods over the full 60,000-sec duration of the simulation runs.

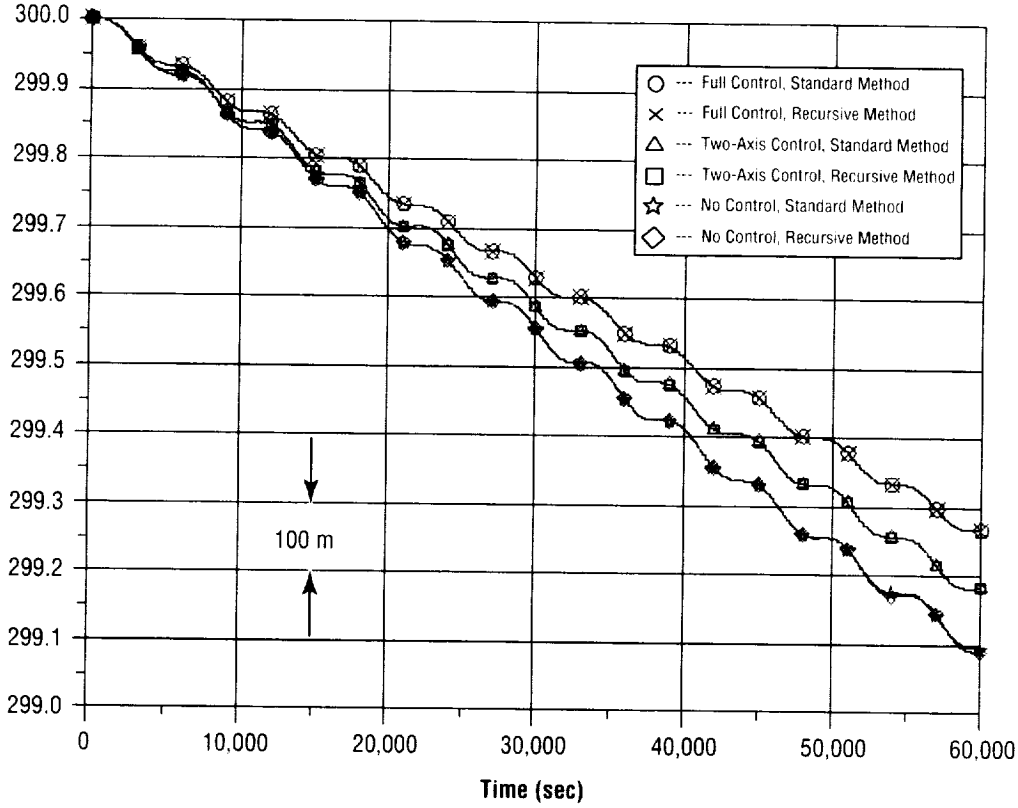


Figure 17. Altitude, case-method array, full run duration (km).

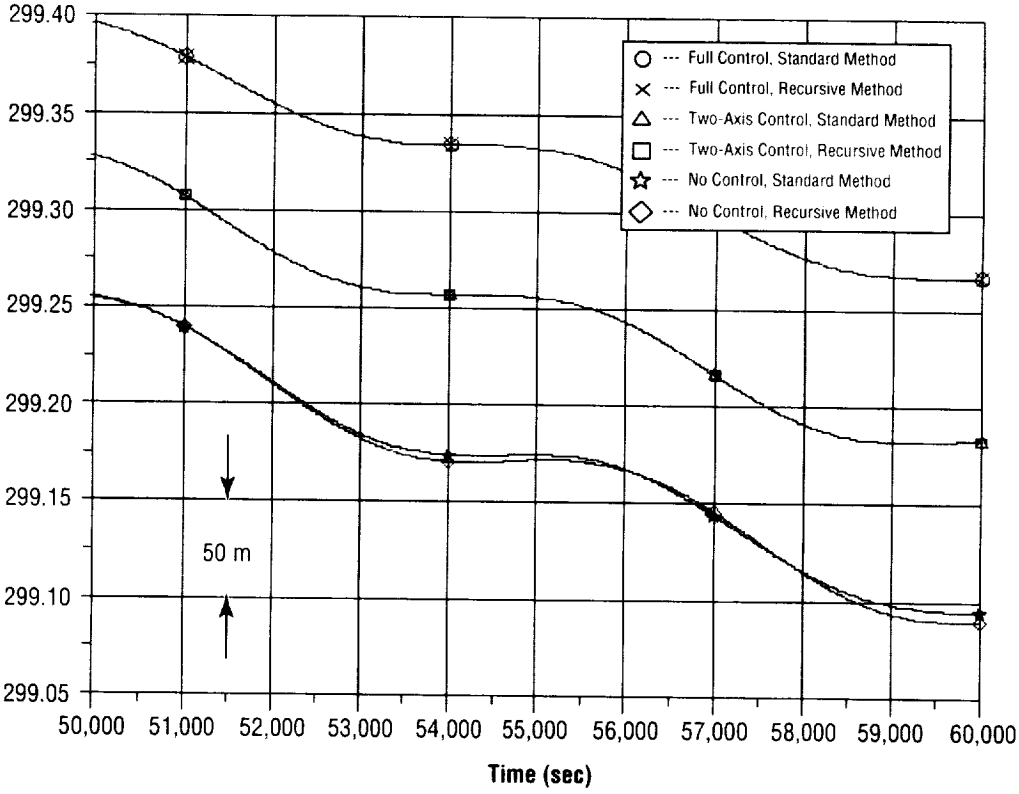


Figure 18. Altitude, case-method array, first time segment (km).

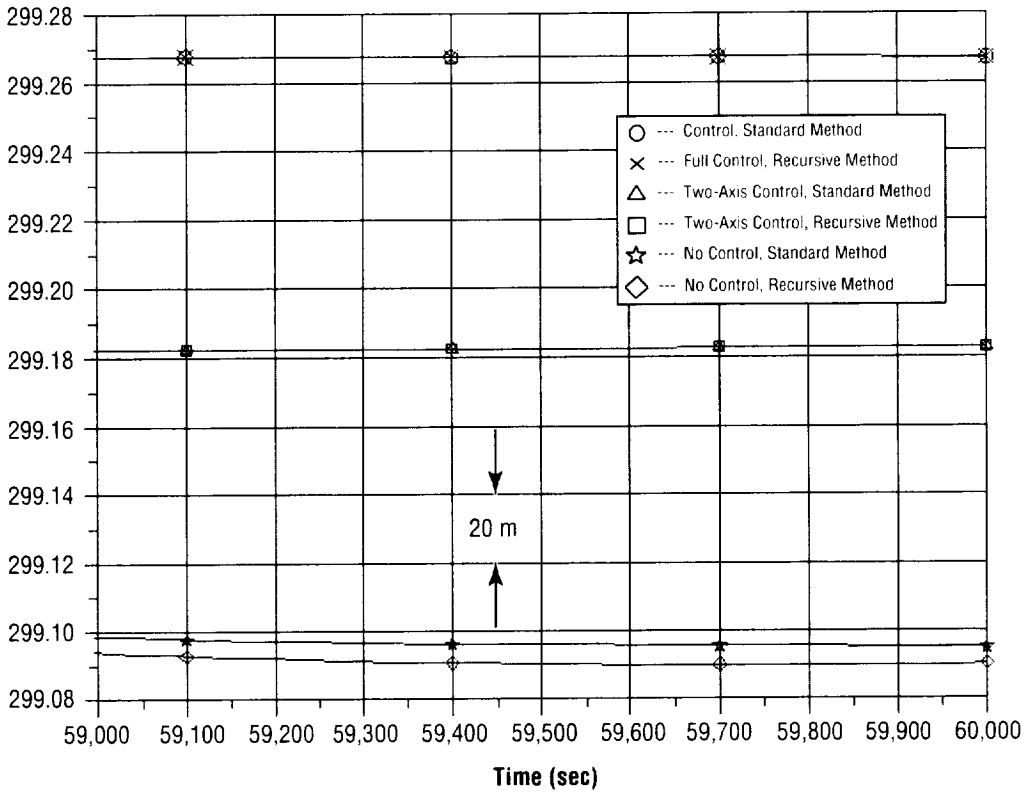


Figure 19. Altitude, case-method array, second time segment (km).

8. PERFORMANCE

The numerical results presented in figures 8–19 and in appendix A very clearly confirm that coupling of discipline-oriented subsimulations by recursive feedback can be used successfully as an architecture and solution approach to simulation of the example system. In cases 1 and 2, 60,000 sec of run time is achieved with no significant deviation of results. In case 3, deviation begins at approximately 45,000 sec. The earlier deviation of case 3 is attributed primarily to linear interpolation of aerodynamic forces and torques over the subsimulation integration step size in the recursive (RF–C and RF–D) implementations, while aerodynamic computations are made at points internal to the integration interval in the unified (RK–U) implementation. Improvement could likely be found through a more sophisticated interpolation method, a shorter convergence interval, or an increased number of points per convergence interval for signal definition. It must be remembered, however, that eventual disagreement of dynamic simulations driven by different numerical processes is always expected.

The results also confirm that the location and distribution of coupling functions, while partially defined by the fundamental nature of the subsystem interactions, can be part of the simulation design and can have a significant affect on performance. For the current example, the convergence interval length (0.4 sec) and the number of points per convergence interval (five) are the same for both recursive approaches, and the accompanying subsimulations as well as the conventional simulation all use the same integration step size (0.1 sec). Convergence error tolerances were in the range of 10^{-12} to 10^{-14} for the recursive methods. Typical reported processor utilization times for the RF–C, and RF–D simulations are approximately 1,200 and 800 sec, respectively, while the RK–U time is approximately 200 sec.

As mentioned earlier, the potential parallel computation advantage of the RF–C simulation is lost because of the particular single-computer FORTRAN implementation. Given that fact, the RF–D (fig. 6) simulation is understandably faster than the RF–C (fig. 4) because the amount of data being processed and checked to confirm or deny convergence at the recursive transfer point is significantly less. The speed of the RF–C simulation would be increased if the subsimulations were actually run in parallel. The possibility for running subsimulations in parallel does not exist for the RF–D simulation because the process design requires them to be run sequentially. In either recursive method, output from the last stage of the process is used to compare against the previous stage for confirmation of convergence. In the conventional method, no active control of error is currently present, and inclusion of any such mechanism would require additional computation; conversely, predetermination of the number of recursion stages and elimination of the convergence check could significantly reduce the computation load for either recursive method. For the current example, the maximum number of recursion stages is typically three, one of which could arguably be said to serve no purpose other than to confirm satisfaction of error tolerances. As the number of recursion stages increases, the extra time required for the confirmation stage becomes less significant in a relative sense, but elimination of the checking process entirely would save time at every stage.

None of the current simulations were constructed with emphasis on efficiency. After correctness, programmer convenience and listing readability were the next higher priority considerations. It is believed that many strengths of recursively coupled subsimulations architecture lie in areas other than computational efficiency and those strengths may sometimes be well afforded at considerable expense.

9. IMMEDIATE FUTURE DEVELOPMENT

More analytical investigation and development are appropriate. Analyses that investigate error sources, error propagation, computational efficiency, and stability are needed. Analyses that address applicability to stochastic systems and discrete data systems are also needed. Further reduction of conservatism of the criterion for convergence derived for systems with algebraic loops (ref. 6) should be explored. The basic concept should be extended to include automatic control of convergence interval duration and other process parameters to improve performance. Implications of central versus distributed or mixed coupling architectures should be considered from viewpoints of algorithm performance and computational resource utilization.

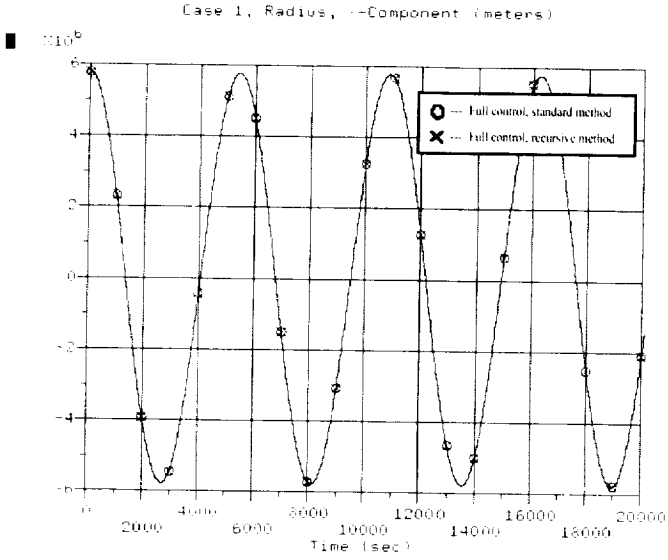
To facilitate a multicomputer implementation, a high-level standard format for digital definition of a vector analog signal segment (function of time) needs to be chosen or created so that signal segments can be transmitted and received throughout the system. In addition, system-level command and control functions must be defined so that standard subsimulation and coupler interfaces can be developed. A single-computer multiprocessor development that takes advantage of parallel computation could advance development of requirements for a multicomputer implementation as well as facilitate a better assessment of performance. Construction of an example involving more than one computer will serve to identify a detailed set of platform- and network-specific requirements in data systems and software terms; it will also serve to explicitly demonstrate technical integrity for a multiplatform application.

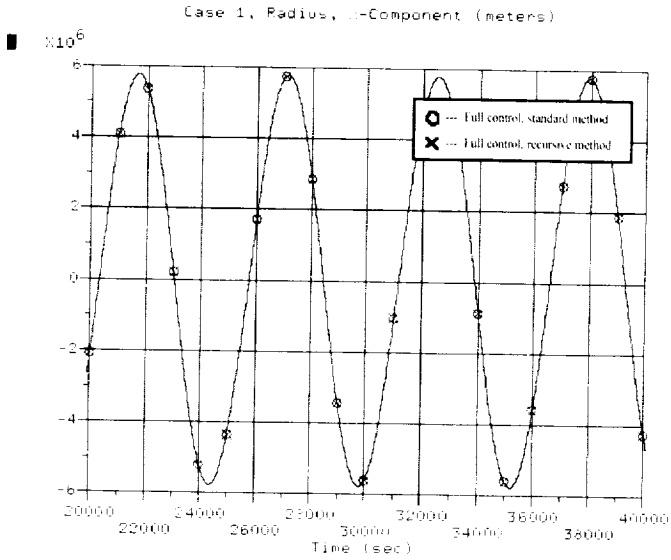
10. CONCLUSIONS AND RECOMMENDATIONS

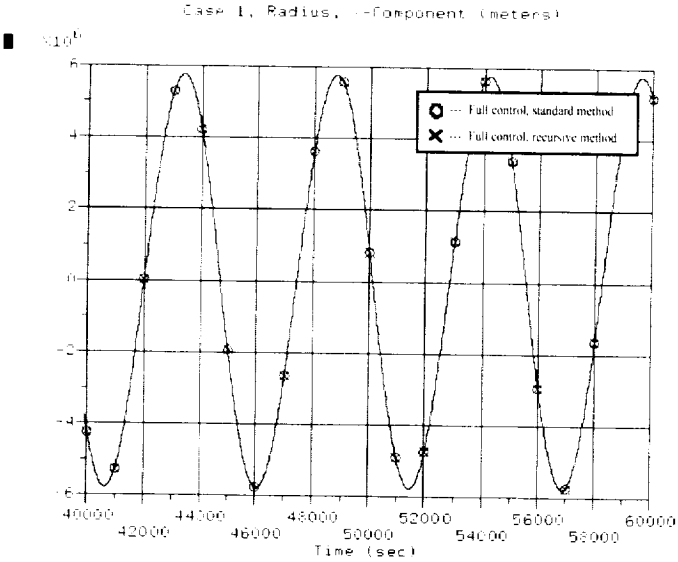
An example nonlinear, continuous simulation of satellite motion has been successfully constructed using two variations of recursive feedback to couple three separate subsimulations of orbit dynamics, attitude dynamics and control, and aerodynamics. Results have been verified by direct comparison with a conventionally constructed simulation of the same system. Each subsimulation deals with one engineering discipline, and appropriate interactions are implemented recursively at the system level. The subsimulations can, in principle, be run separately on remote and dissimilar platforms while coupling is implemented by way of network. Depending on the system-level design configuration, it is possible to run the subsimulations in parallel. The coupling signals represent physically identifiable variables associated with physical interfaces of the subsystems. The example employs three subsimulations; the method framework accommodates n. Clearly, further investigation of recursively coupled subsimulations as a systems simulation architecture is warranted. It is recommended that the present effort be expanded to include all of the future development areas mentioned above, and that longer range plans should include a pilot multisubsystem, multidiscipline, multiplatform development of a complexity level that demonstrates effectiveness and efficiency through collaboration.

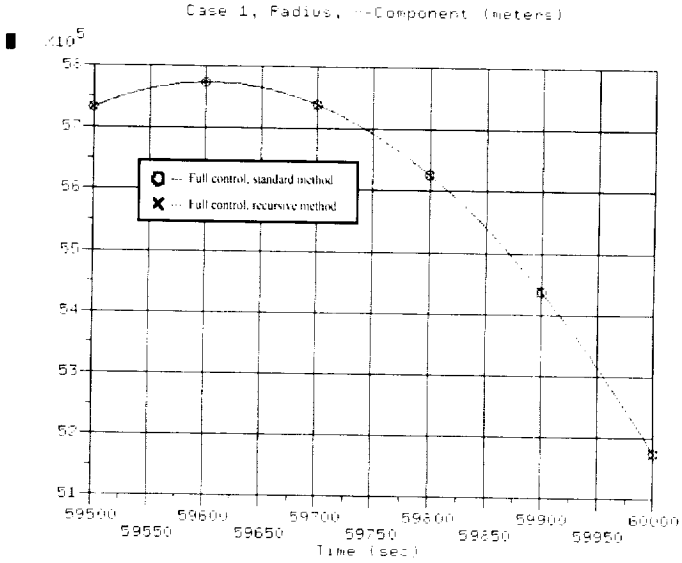
APPENDIX A—RESPONSE PLOTS

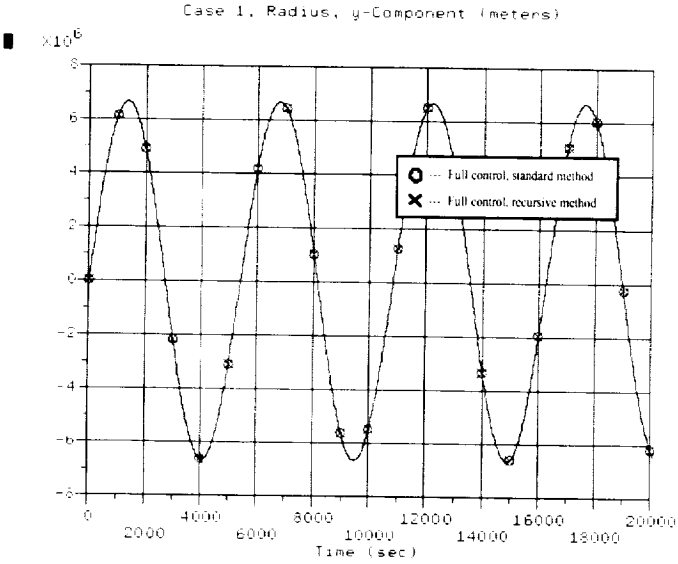
A.1 Response Plots for Case 1 (Full Three-Axis Inertial Attitude Control)

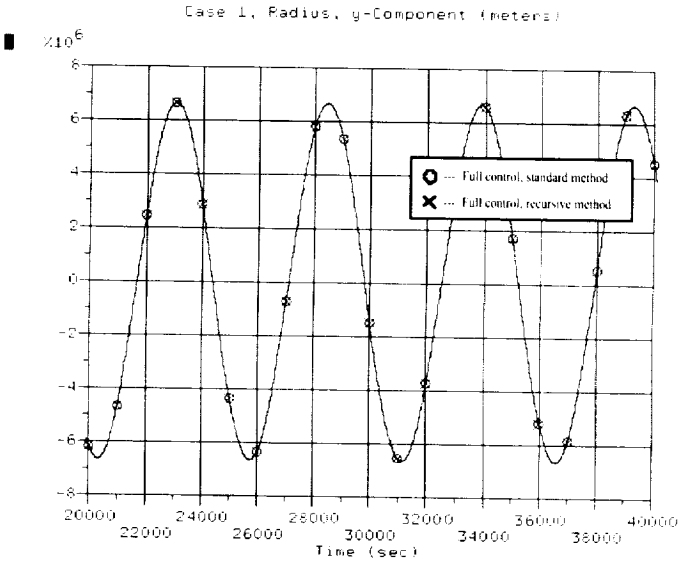

Case 1, Radius, x-Component (meters)
x10^6
Full control, standard method
Full control, recursive method
Time (sec)

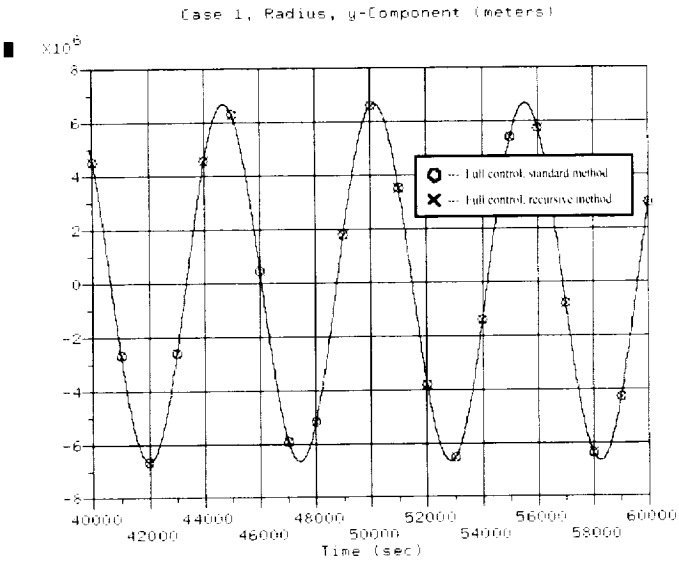

Case 1, Radius, x-Component (meters)
x10^6
Full control, standard method
Full control, recursive method
Time (sec)

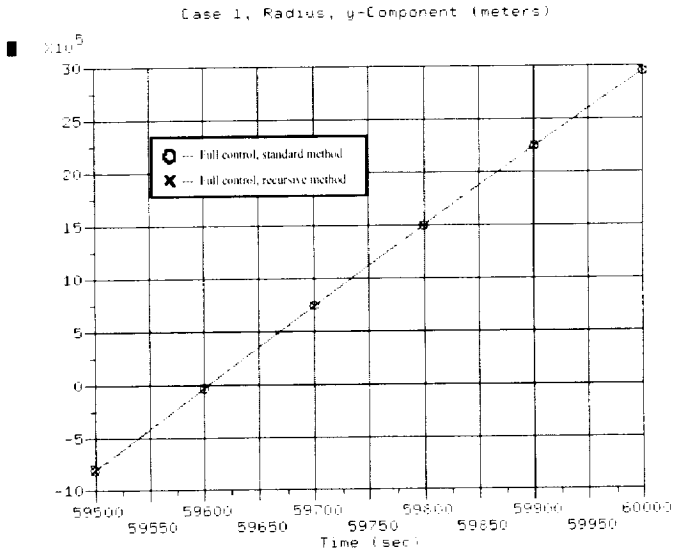

Case 1, Radius, z-Component (meters)
x10^6
Full control, standard method
Full control, recursive method
Time (sec)

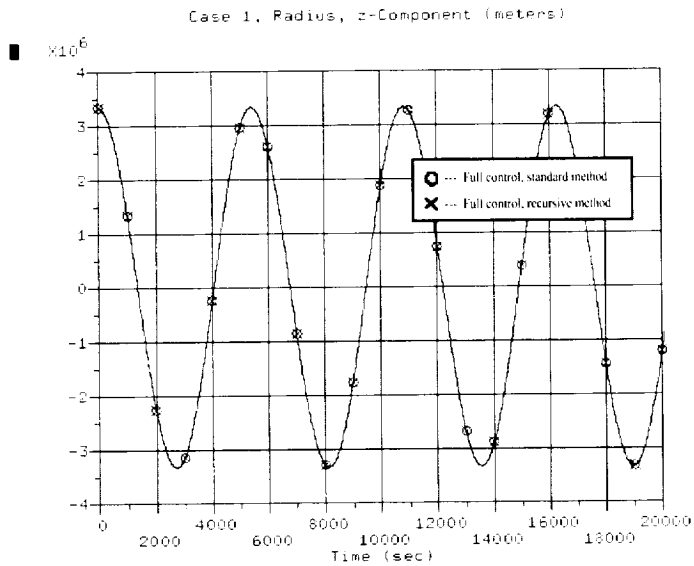

Case 1, Radius, z-Component (meters)
x10^5
Full control, standard method
Full control, recursive method
Time (sec)

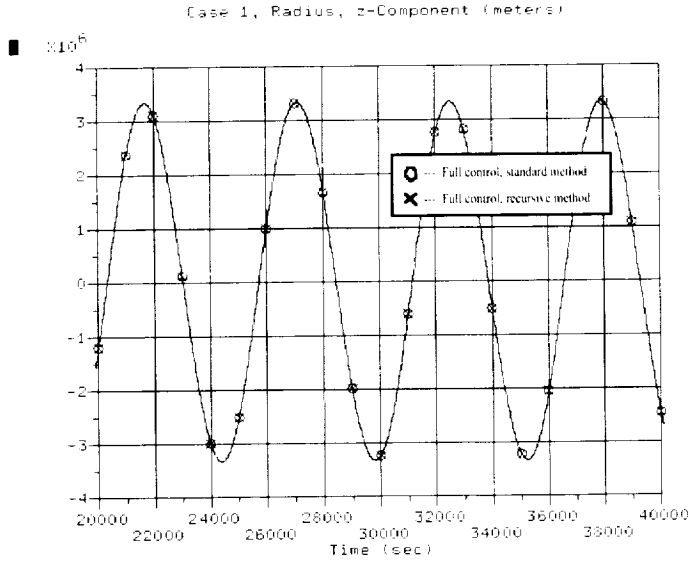

Case 1, Radius, y-Component (meters)
x10^6
Full control, standard method
Full control, recursive method
Time (sec)

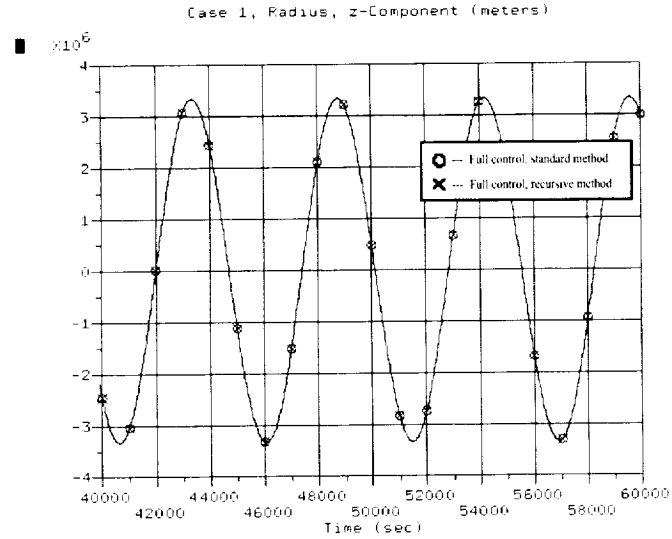

Case 1, Radius, y-Component (meters)
x10^6
Full control, standard method
Full control, recursive method
Time (sec)

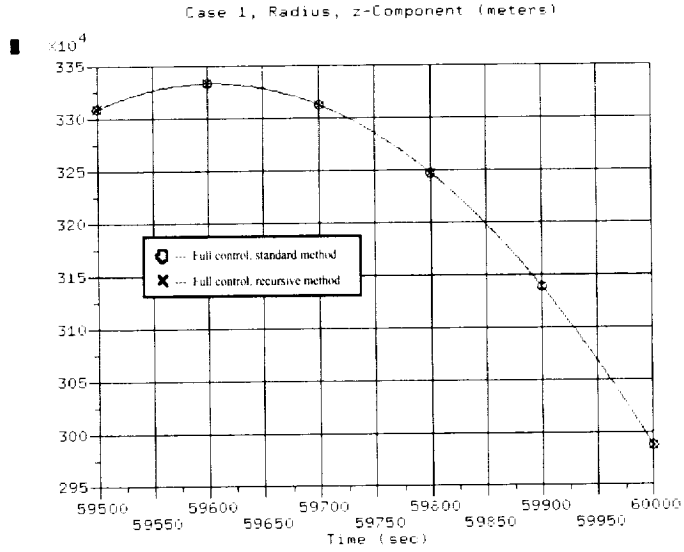

Case 1, Radius, y-Component (meters)
X10^6
8
6
4
2
0
-2
-4
-6
-8
40000
42000
44000
46000
48000
50000
52000
54000
56000
58000
60000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

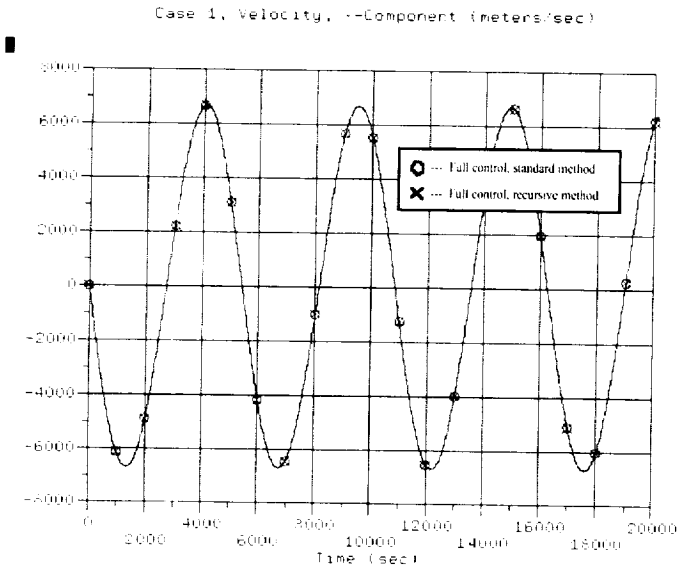

Case 1, Radius, y-Component (meters)
X10^5
30
25
20
15
10
5
0
-5
-10
59500
59550
59600
59650
59700
59750
59800
59850
59900
59950
60000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

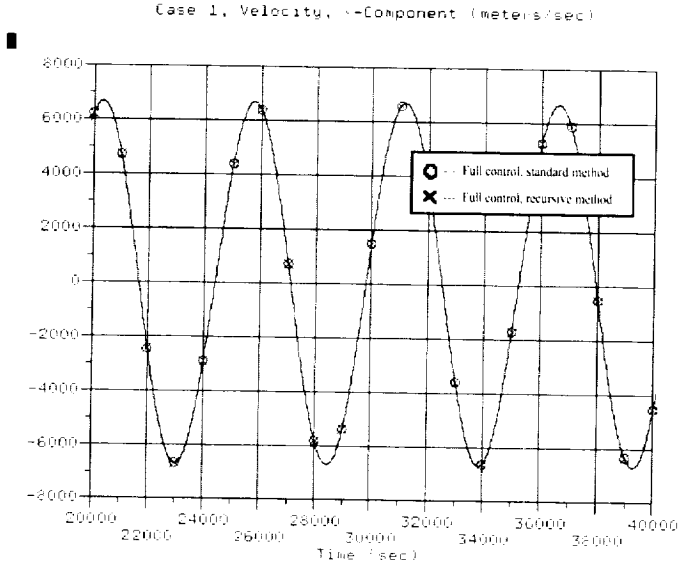

Case 1, Radius, z-Component (meters)
X10^6
4
3
2
1
0
-1
-2
-3
-4
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

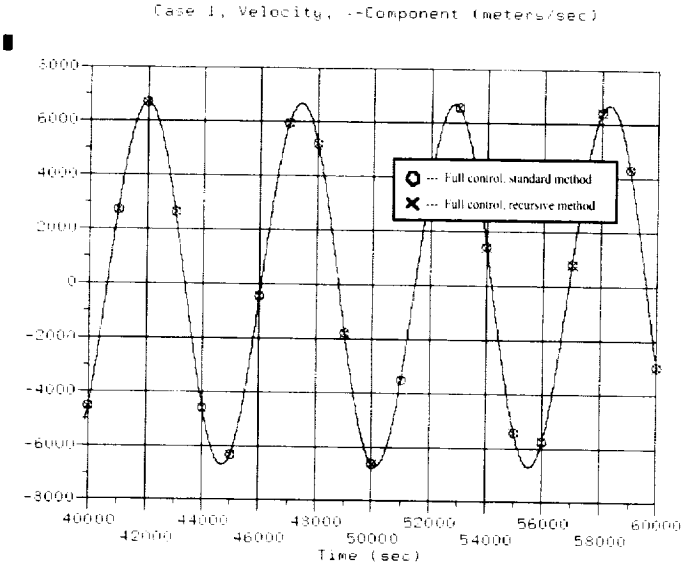

Case 1, Radius, z-Component (meters)
X10^6
4
3
2
1
0
-1
-2
-3
-4
20000
22000
24000
26000
28000
30000
32000
34000
36000
38000
40000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

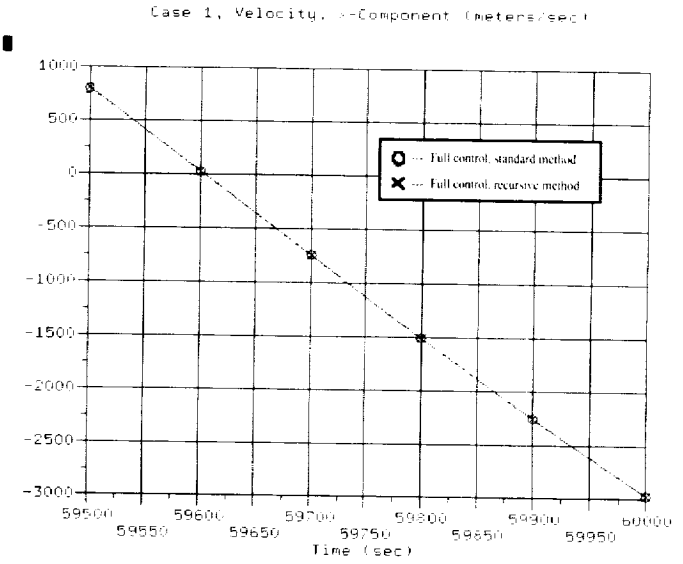

Case 1, Radius, z-Component (meters)
X10^6
4
3
2
1
0
-1
-2
-3
-4
40000
42000
44000
46000
48000
50000
52000
54000
56000
58000
60000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

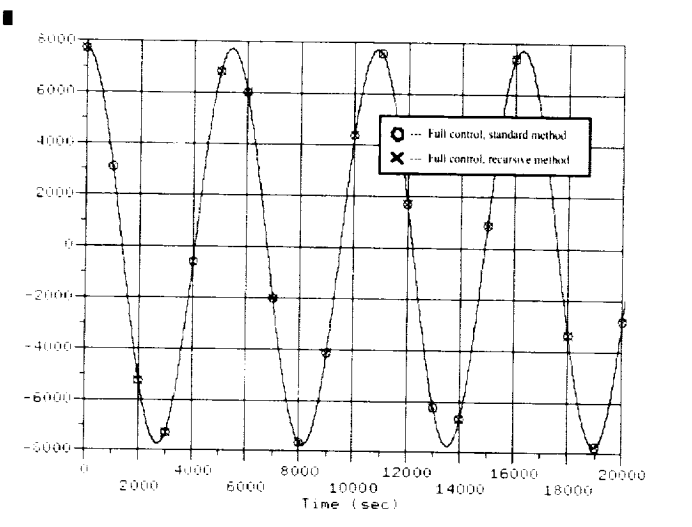

Case 1, Radius, z-Component (meters)
X10^4
335
330
325
320
315
310
305
300
295
59500
59550
59600
59650
59700
59750
59800
59850
59900
59950
60000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

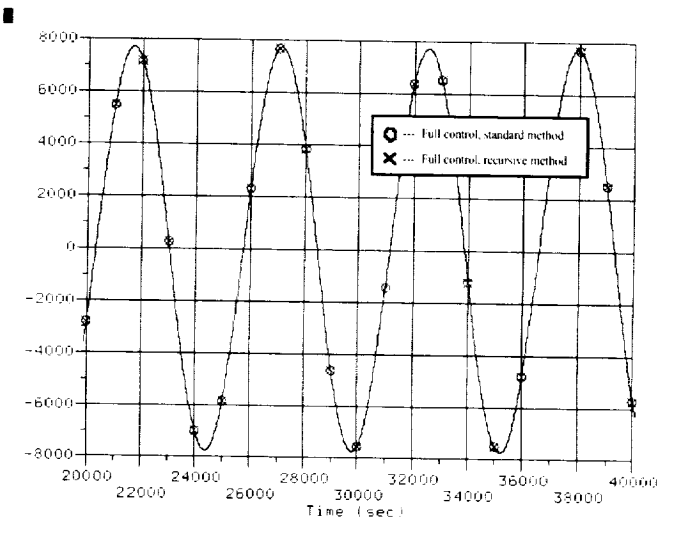

Case 1, Velocity, x-Component (meters/sec)
Full control, standard method
Full control, recursive method
Time (sec)

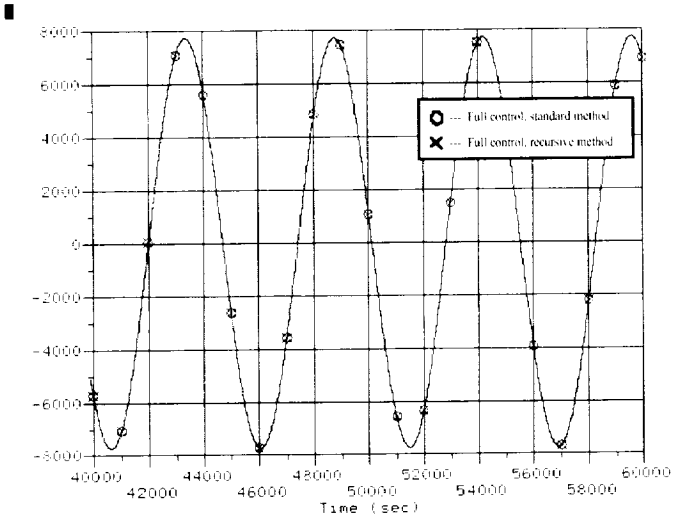

Case 1, Velocity, x-Component (meters/sec)
Full control, standard method
Full control, recursive method
Time (sec)

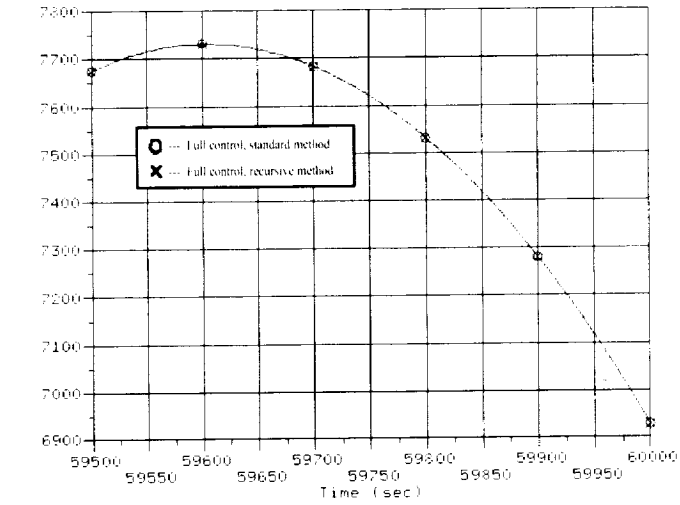

Case 1, Velocity, z-Component (meters/sec)
Full control, standard method
Full control, recursive method
Time (sec)

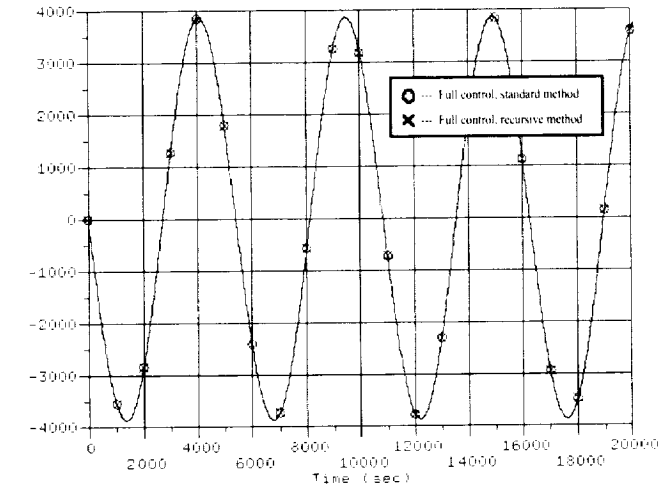

Case 1, Velocity, z-Component (meters/sec)
Full control, standard method
Full control, recursive method
Time (sec)

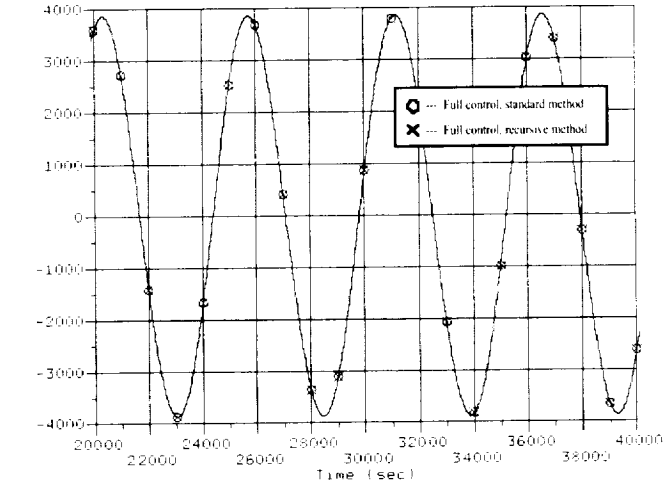

Case 1, Velocity, y-Component (meters/sec)
Full control, standard method
Full control, recursive method
Time (sec)

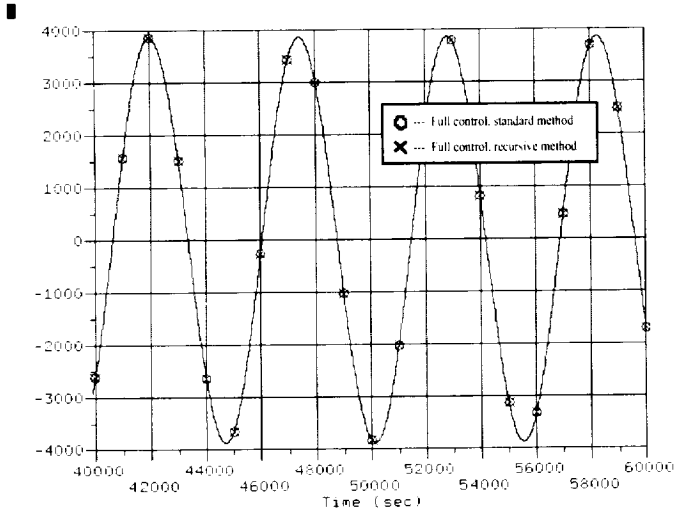

Case 1, Velocity, y-Component (meters/sec)
Full control, standard method
Full control, recursive method
Time (sec)

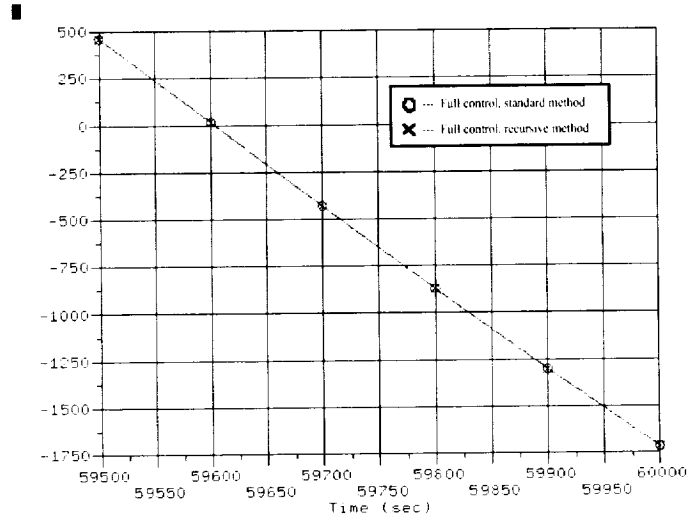

Case 1, Velocity, y-Component (meters/sec)

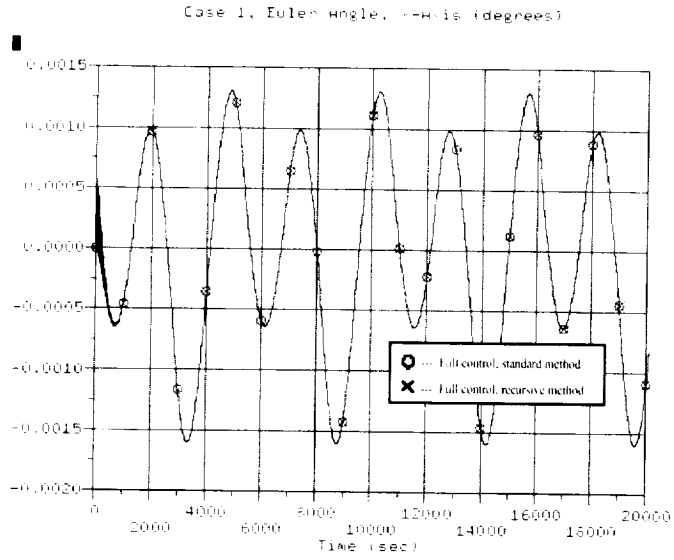

Case 1, Velocity, y-Component (meters/sec)

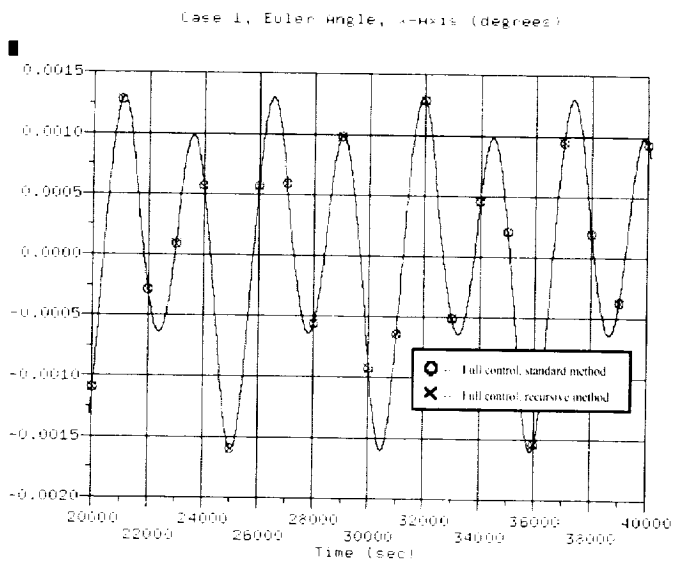

Case 1, Velocity, z-Component (meters/sec)

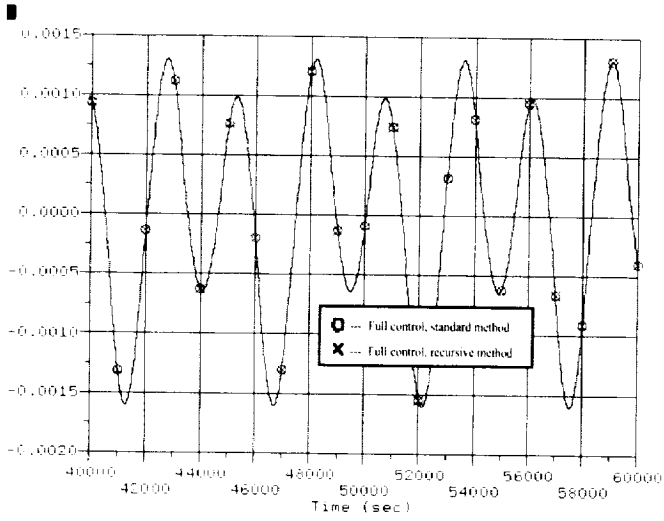

Case 1, Velocity, z-Component (meters/sec)

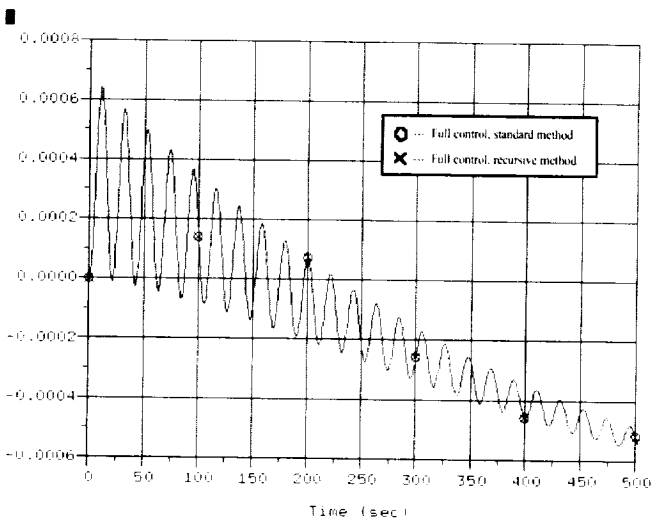

Case 1, Velocity, z-Component (meters/sec)

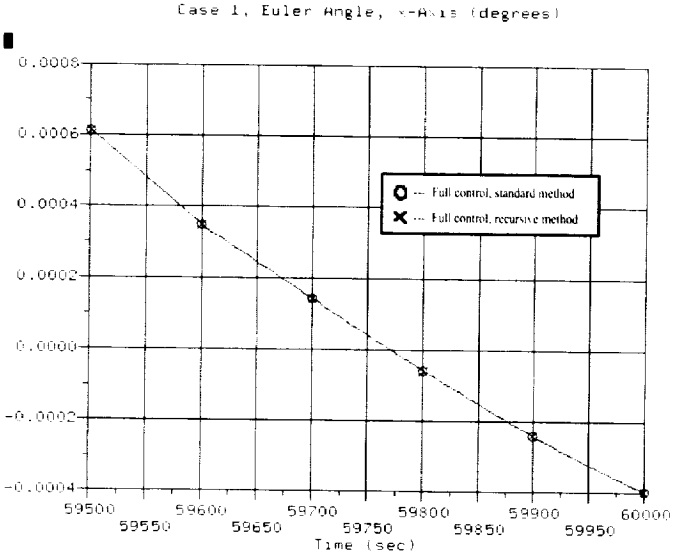

Case 1, Velocity, z-Component (meters/sec)

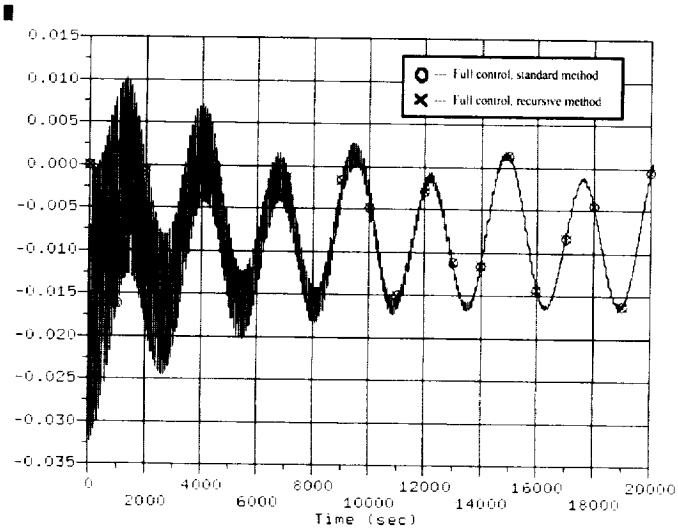

Case 1, Euler Angle, x-Axis (degrees)
0.0015
0.0010
0.0005
0.0000
-0.0005
-0.0010
-0.0015
-0.0020
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

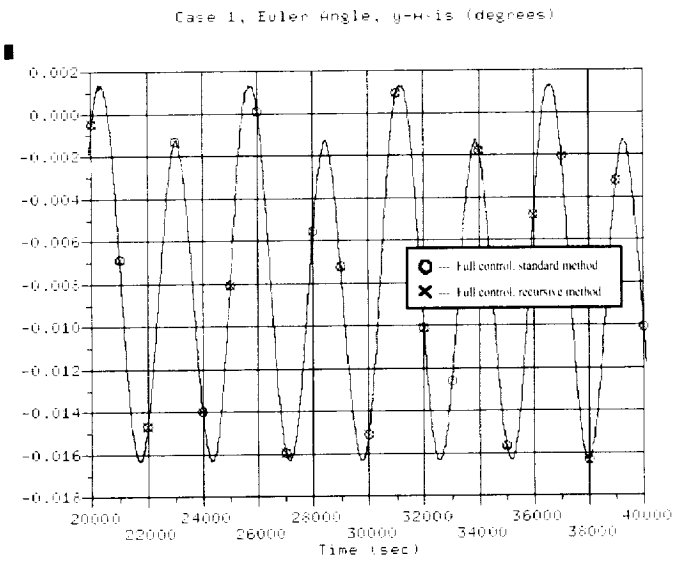

Case 1, Euler Angle, x-Axis (degrees)
0.0015
0.0010
0.0005
0.0000
-0.0005
-0.0010
-0.0015
-0.0020
20000
22000
24000
26000
28000
30000
32000
34000
36000
38000
40000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

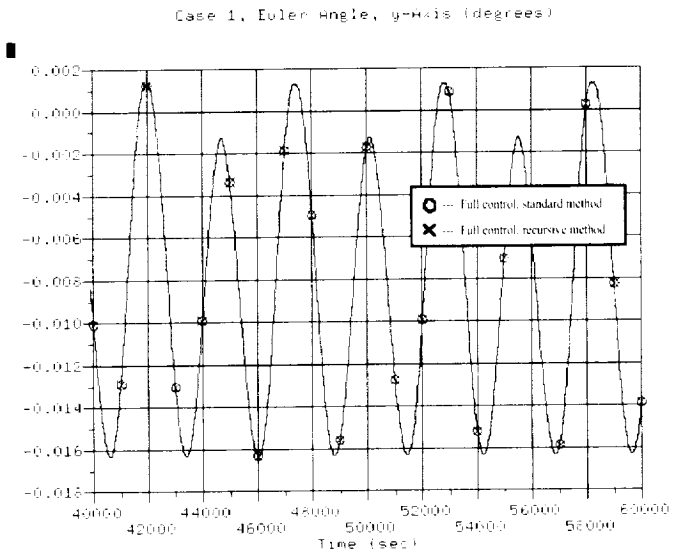

Case 1, Euler Angle, x-Axis (degrees)
0.0015
0.0010
0.0005
0.0000
-0.0005
-0.0010
-0.0015
-0.0020
40000
42000
44000
46000
48000
50000
52000
54000
56000
58000
60000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

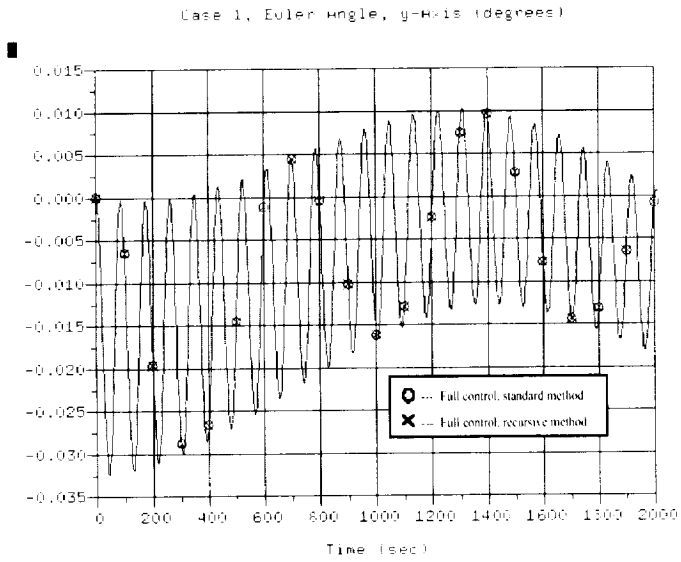

Case 1, Euler Angle, x-Axis (degrees)
0.0008
0.0006
0.0004
0.0002
0.0000
-0.0002
-0.0004
-0.0006
0
50
100
150
200
250
300
350
400
450
500
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

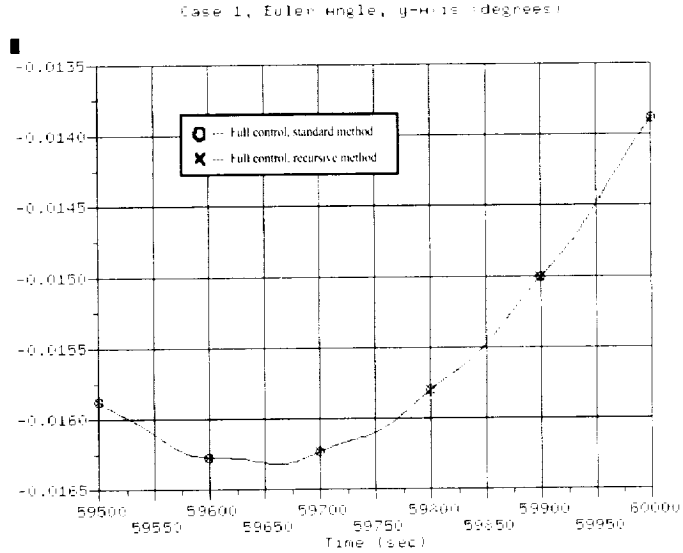

Case 1, Euler Angle, x-Axis (degrees)
0.0008
0.0006
0.0004
0.0002
0.0000
-0.0002
-0.0004
59500
59550
59600
59650
59700
59750
59800
59850
59900
59950
60000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

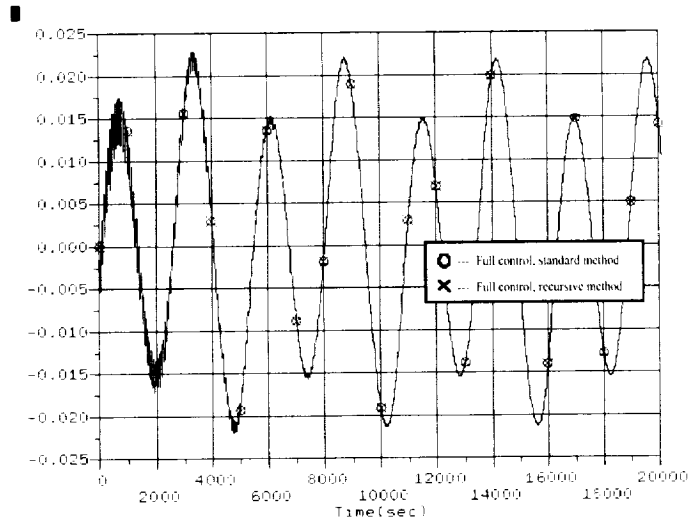

Case 1, Euler Angle, y-Axis (degrees)
0.015
0.010
0.005
0.000
-0.005
-0.010
-0.015
-0.020
-0.025
-0.030
-0.035
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

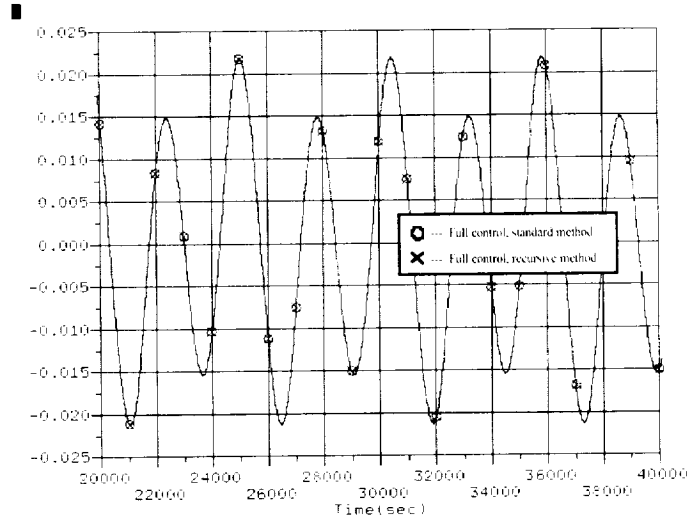

Case 1, Euler Angle, y-Axis (degrees)
0.002
0.000
-0.002
-0.004
-0.006
-0.008
-0.010
-0.012
-0.014
-0.016
-0.018
20000
22000
24000
26000
28000
30000
32000
34000
36000
38000
40000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

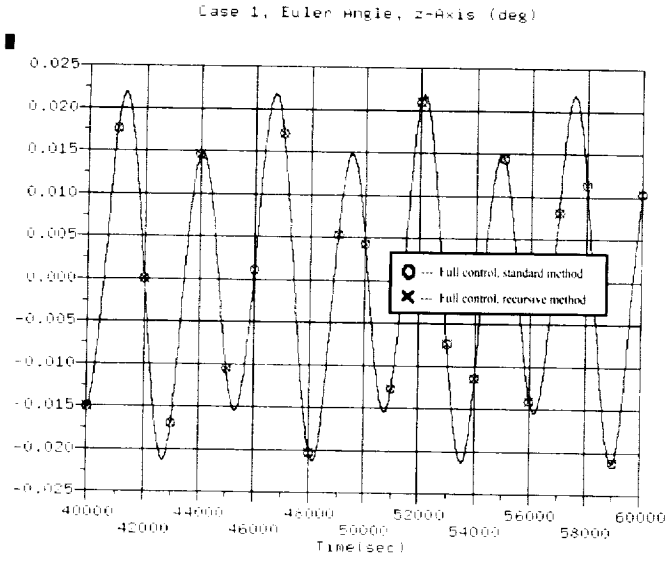

Case 1, Euler Angle, y-Axis (degrees)
0.002
0.000
-0.002
-0.004
-0.006
-0.008
-0.010
-0.012
-0.014
-0.016
-0.018
40000
42000
44000
46000
48000
50000
52000
54000
56000
58000
60000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

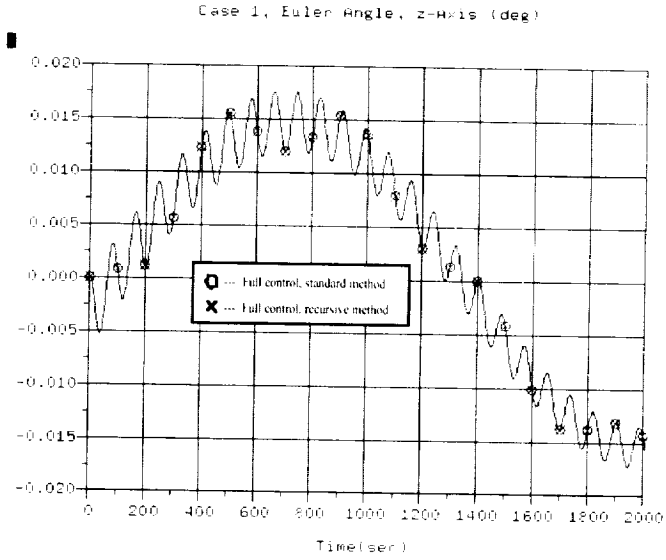

Case 1, Euler Angle, y-Axis (degrees)
0.015
0.010
0.005
0.000
-0.005
-0.010
-0.015
-0.020
-0.025
-0.030
-0.035
0
200
400
600
800
1000
1200
1400
1600
1800
2000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

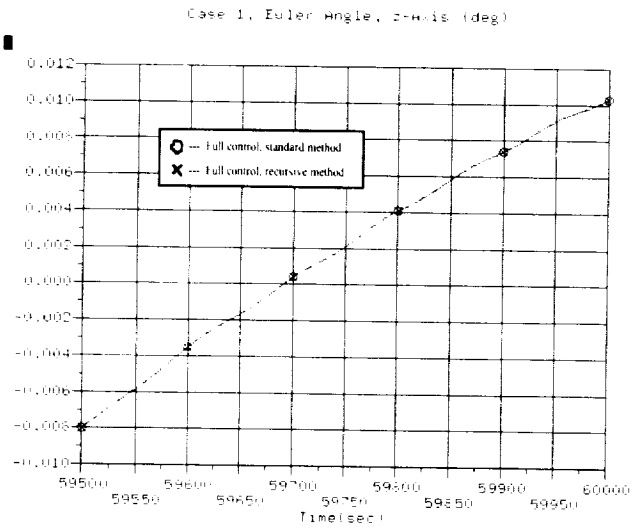

Case 1, Euler Angle, y-Axis (degrees)
-0.0135
-0.0140
-0.0145
-0.0150
-0.0155
-0.0160
-0.0165
59500
59550
59600
59650
59700
59750
59800
59850
59900
59950
60000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

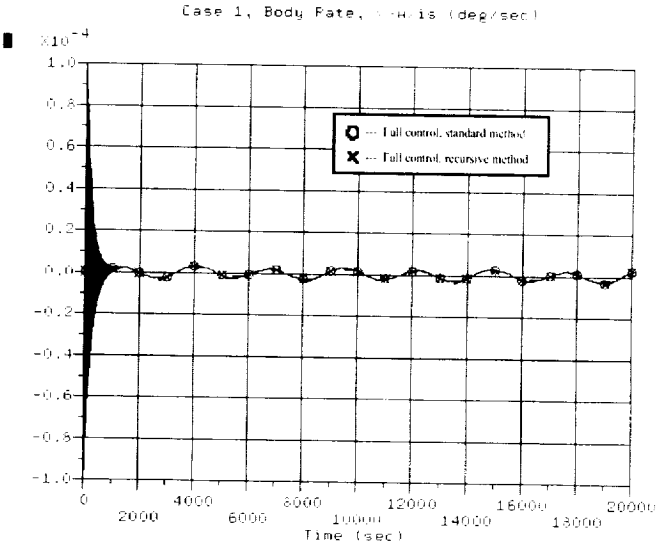

Case 1, Euler Angle, z-Axis (deg)
0.025
0.020
0.015
0.010
0.005
0.000
-0.005
-0.010
-0.015
-0.020
-0.025
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Time(sec)
O --- Full control, standard method
X --- Full control, recursive method

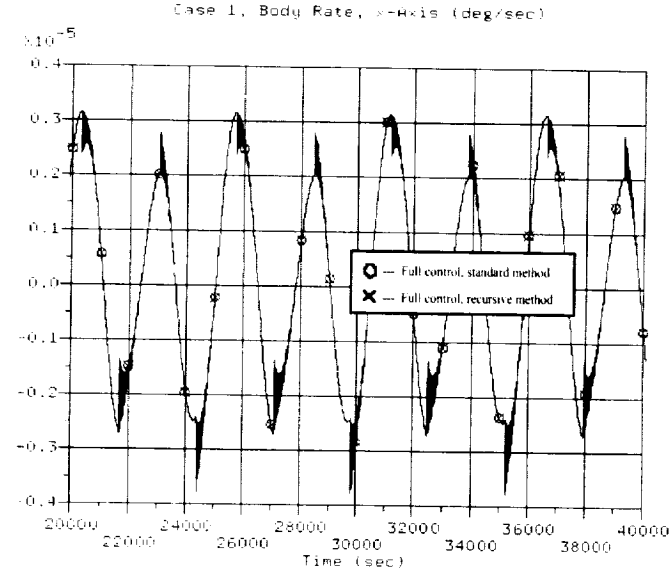

Case 1, Euler Angle, z-Axis (deg)
0.025
0.020
0.015
0.010
0.005
0.000
-0.005
-0.010
-0.015
-0.020
-0.025
20000
22000
24000
26000
28000
30000
32000
34000
36000
38000
40000
Time(sec)
O --- Full control, standard method
X --- Full control, recursive method

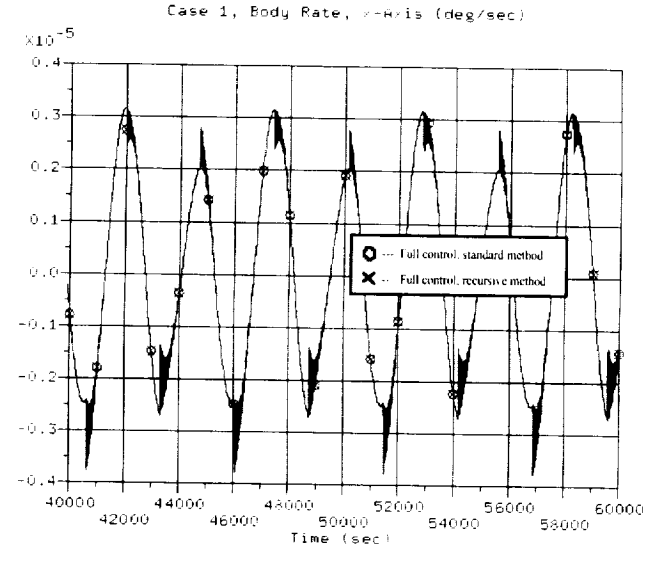

Case 1, Euler Angle, z-Axis (deg)
O --- Full control, standard method
X --- Full control, recursive method
Time(sec)

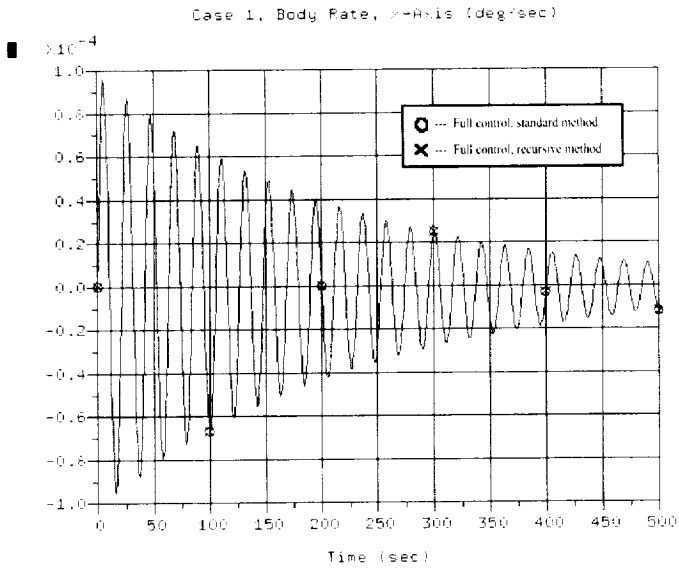

Case 1, Euler Angle, z-Axis (deg)
O --- Full control, standard method
X --- Full control, recursive method
Time(sec)

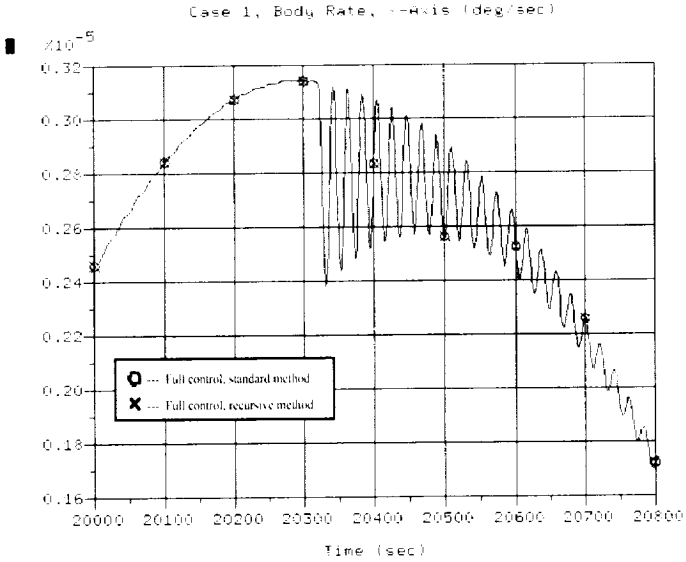

Case 1, Euler Angle, z-Axis (deg)
O --- Full control, standard method
X --- Full control, recursive method
Time(sec)

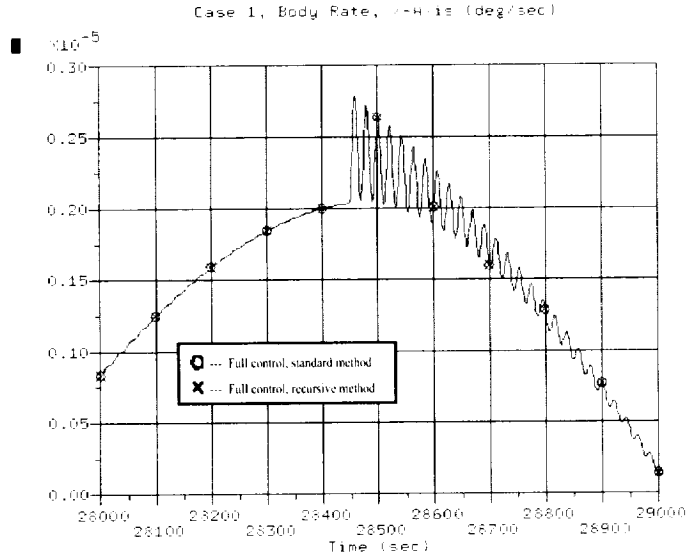

Case 1, Body Rate, x-Axis (deg/sec)
O --- Full control, standard method
X --- Full control, recursive method
Time (sec)

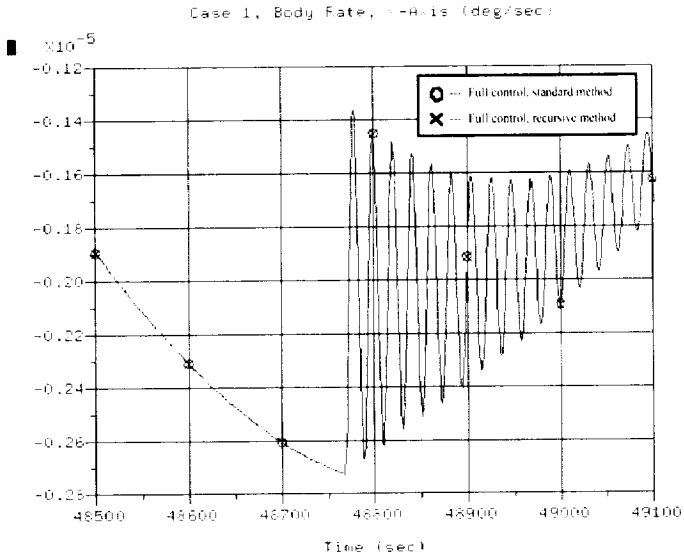

Case 1, Body Rate, x-Axis (deg/sec)
O --- Full control, standard method
X --- Full control, recursive method
Time (sec)

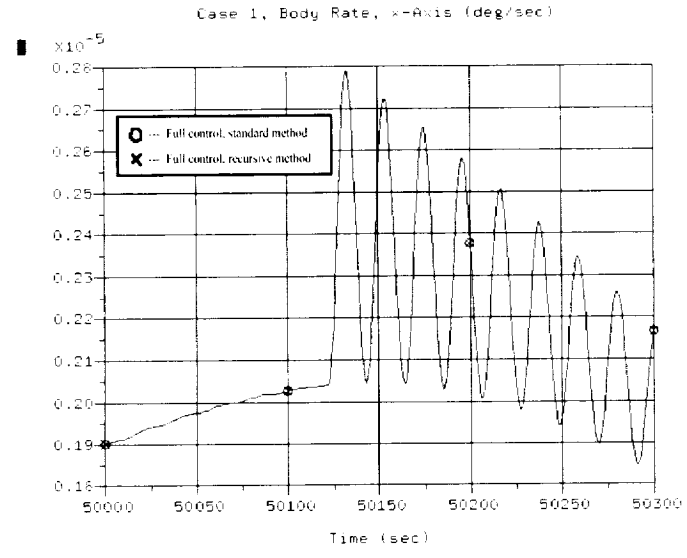

Case 1, Body Rate, x-Axis (deg/sec)
O --- Full control, standard method
X --- Full control, recursive method
Time (sec)

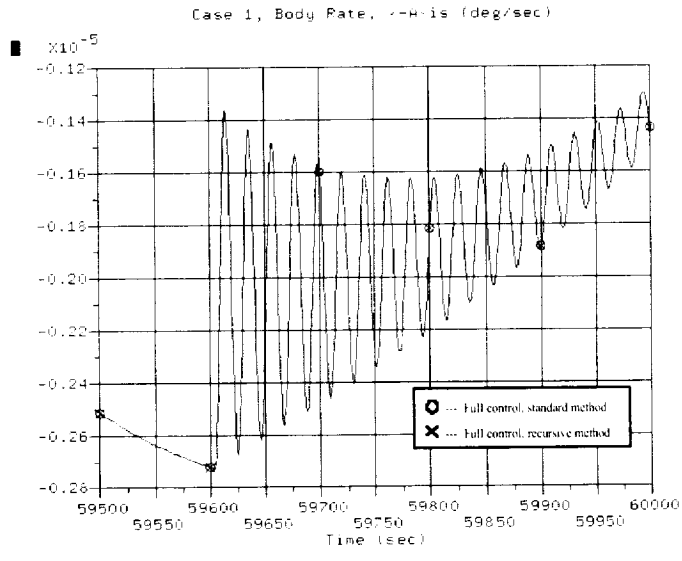

Case 1, Body Rate, x-Axis (deg/sec)
X10^-4
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
-0.4
-0.6
-0.8
-1.0
0
50
100
150
200
250
300
350
400
450
500
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

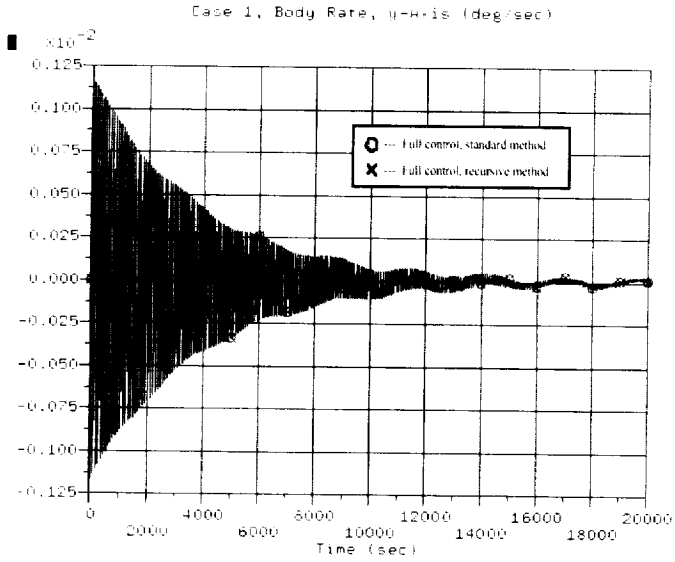

Case 1, Body Rate, x-Axis (deg/sec)
X10^-5
0.32
0.30
0.28
0.26
0.24
0.22
0.20
0.18
0.16
20000
20100
20200
20300
20400
20500
20600
20700
20800
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

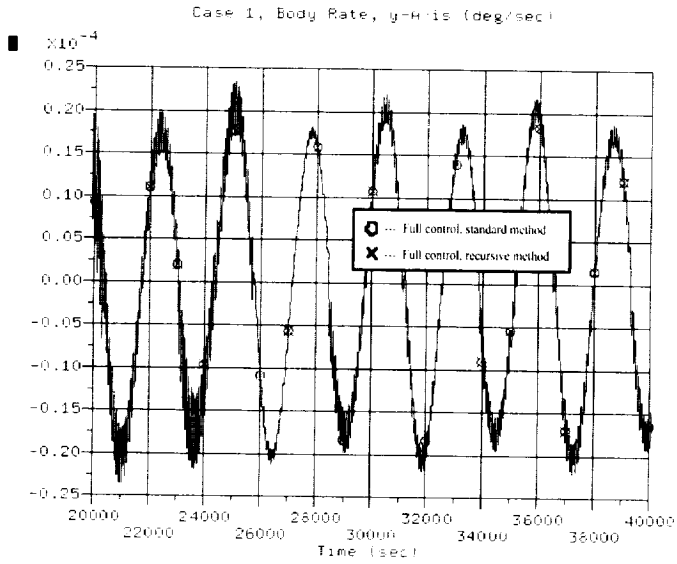

Case 1, Body Rate, x-Axis (deg/sec)
X10^-5
0.30
0.25
0.20
0.15
0.10
0.05
0.00
28000
28100
28200
28300
28400
28500
28600
28700
28800
28900
29000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

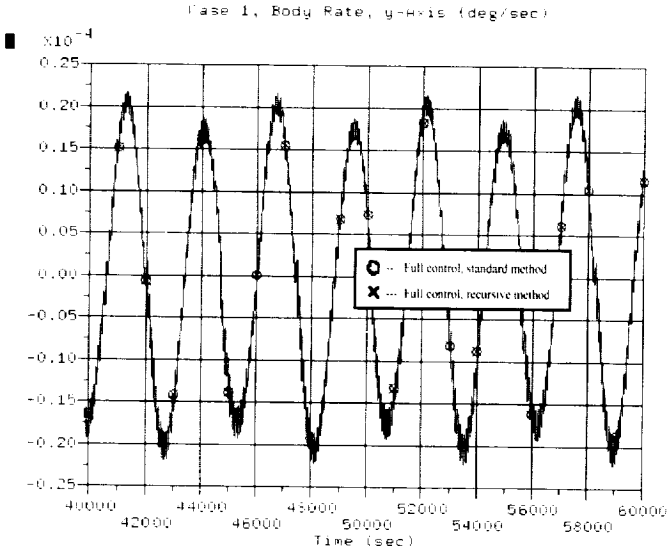

Case 1, Body Rate, x-Axis (deg/sec)
X10^-5
-0.12
-0.14
-0.16
-0.18
-0.20
-0.22
-0.24
-0.26
-0.28
48500
48600
48700
48800
48900
49000
49100
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

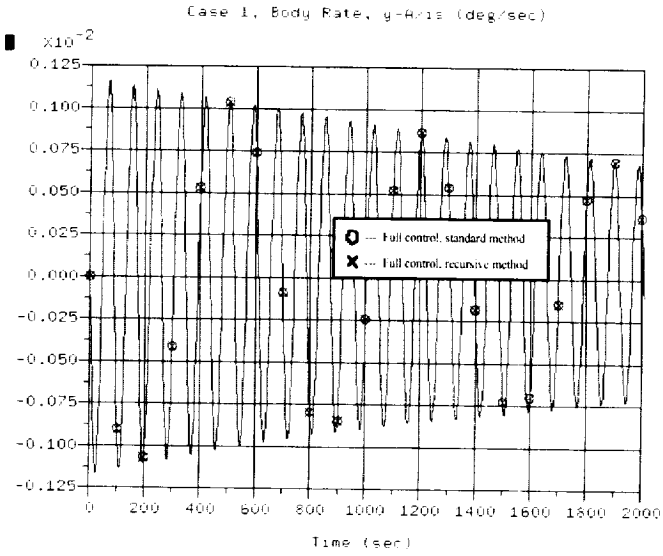

Case 1, Body Rate, x-Axis (deg/sec)
X10^-5
0.28
0.27
0.26
0.25
0.24
0.23
0.22
0.21
0.20
0.19
0.18
50000
50050
50100
50150
50200
50250
50300
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

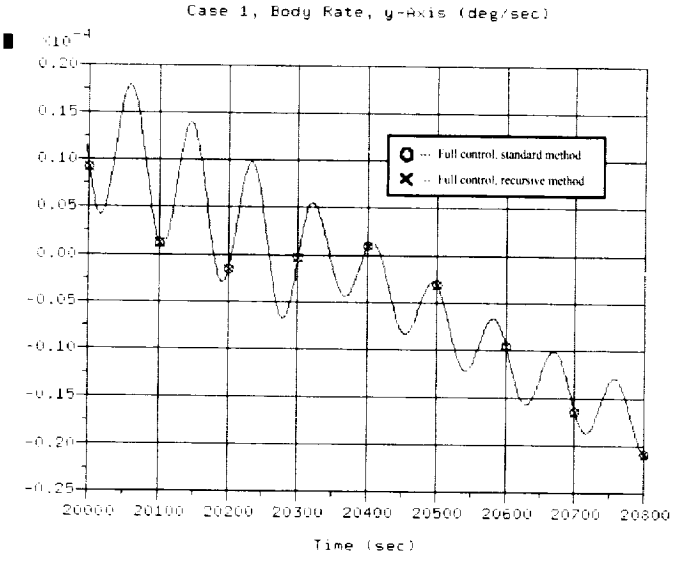

Case 1, Body Rate, x-Axis (deg/sec)
X10^-5
-0.12
-0.14
-0.16
-0.18
-0.20
-0.22
-0.24
-0.26
-0.28
59500
59550
59600
59650
59700
59750
59800
59850
59900
59950
60000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

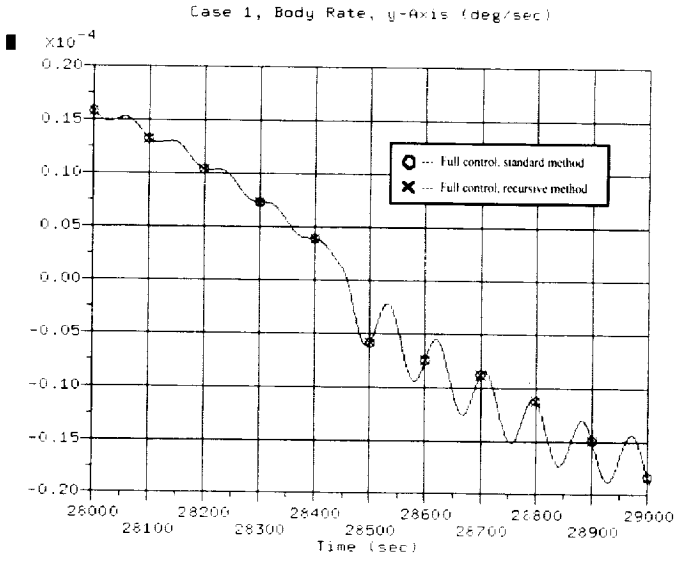

Case 1, Body Rate, y-Axis (deg/sec)
X10^-2
0.125
0.100
0.075
0.050
0.025
0.000
-0.025
-0.050
-0.075
-0.100
-0.125
O --- Full control, standard method
X --- Full control, recursive method
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Time (sec)

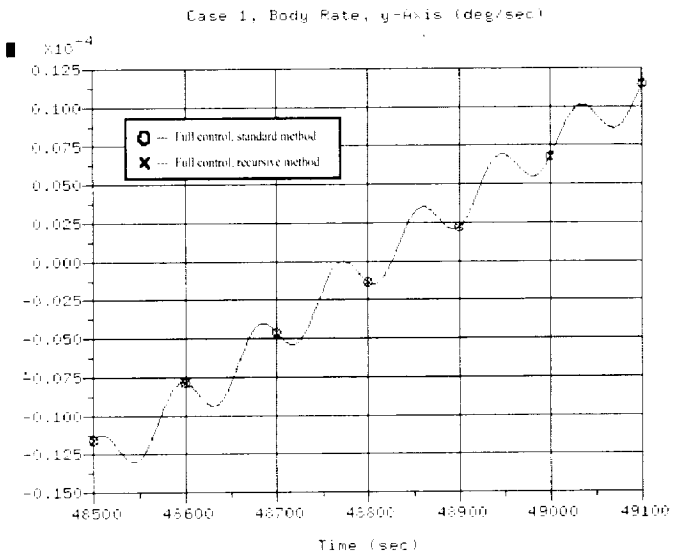

Case 1, Body Rate, y-Axis (deg/sec)
X10^-4
0.25
0.20
0.15
0.10
0.05
0.00
-0.05
-0.10
-0.15
-0.20
-0.25
O --- Full control, standard method
X --- Full control, recursive method
20000
22000
24000
26000
28000
30000
32000
34000
36000
38000
40000
Time (sec)

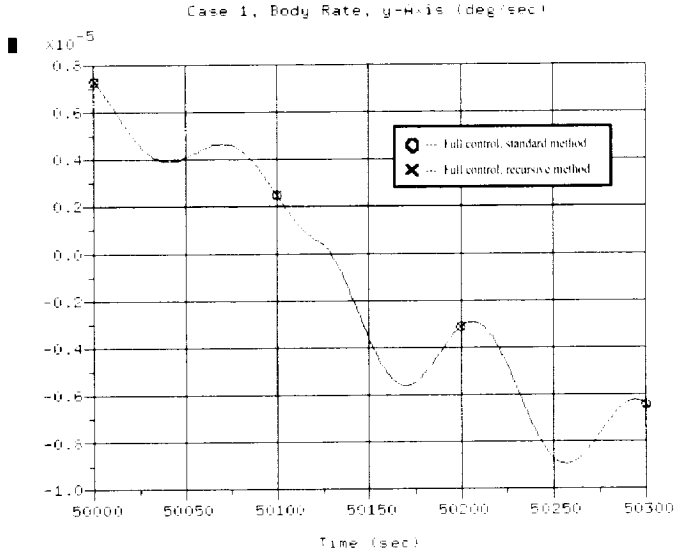

Case 1, Body Rate, y-Axis (deg/sec)
X10^-4
0.25
0.20
0.15
0.10
0.05
0.00
-0.05
-0.10
-0.15
-0.20
-0.25
O --- Full control, standard method
X --- Full control, recursive method
40000
42000
44000
46000
48000
50000
52000
54000
56000
58000
60000
Time (sec)

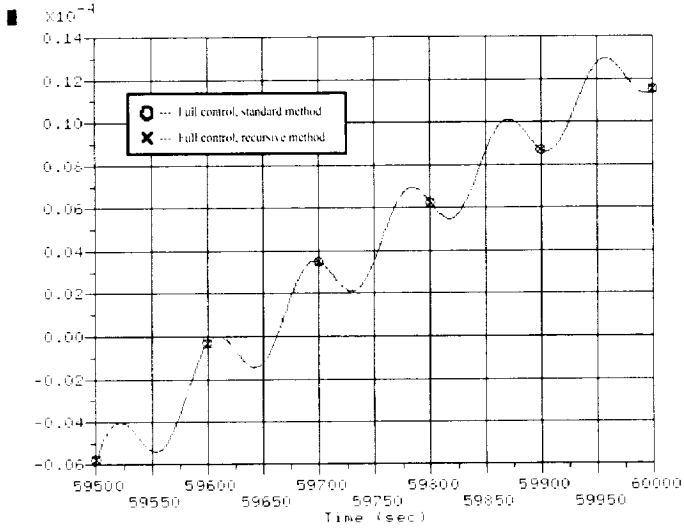

Case 1, Body Rate, y-Axis (deg/sec)
X10^-2
0.125
0.100
0.075
0.050
0.025
0.000
-0.025
-0.050
-0.075
-0.100
-0.125
O --- Full control, standard method
X --- Full control, recursive method
0
200
400
600
800
1000
1200
1400
1600
1800
2000
Time (sec)

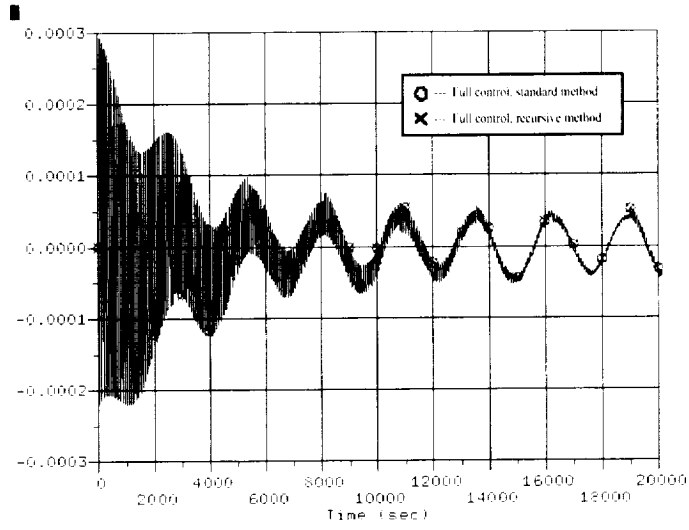

Case 1, Body Rate, y-Axis (deg/sec)
X10^-4
0.20
0.15
0.10
0.05
0.00
-0.05
-0.10
-0.15
-0.20
-0.25
O --- Full control, standard method
X --- Full control, recursive method
20000
20100
20200
20300
20400
20500
20600
20700
20800
Time (sec)

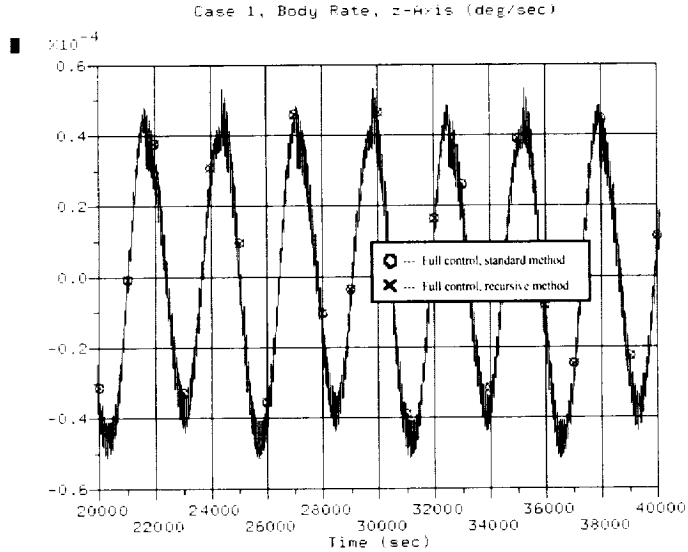

Case 1, Body Rate, y-Axis (deg/sec)
X10^-4
0.20
0.15
0.10
0.05
0.00
-0.05
-0.10
-0.15
-0.20
O --- Full control, standard method
X --- Full control, recursive method
28000
28100
28200
28300
28400
28500
28600
28700
28800
28900
29000
Time (sec)

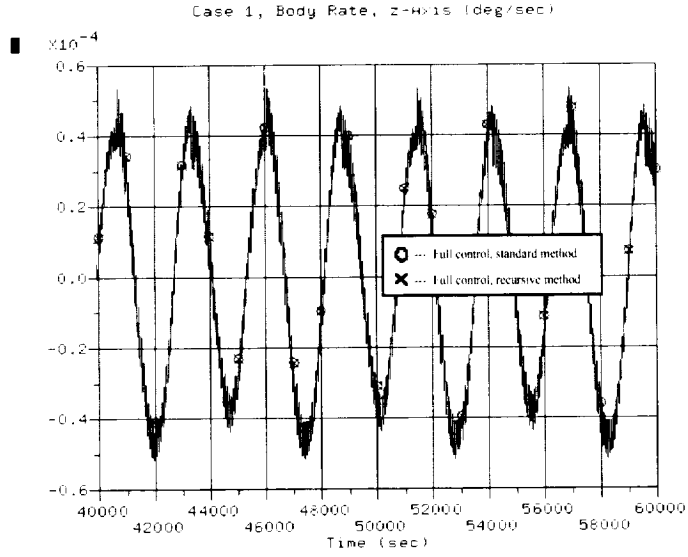

Case 1, Body Rate, y-Axis (deg/sec)
x10^-4
0.125
0.100
0.075
0.050
0.025
0.000
-0.025
-0.050
-0.075
-0.100
-0.125
-0.150
48500
48600
48700
48800
48900
49000
49100
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

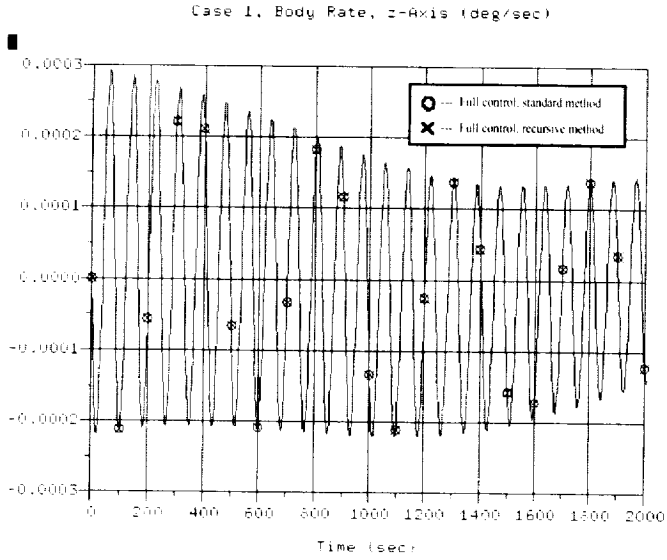

Case 1, Body Rate, y-Axis (deg/sec)
x10^-5
0.8
0.6
0.4
0.2
0.0
-0.2
-0.4
-0.6
-0.8
-1.0
50000
50050
50100
50150
50200
50250
50300
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

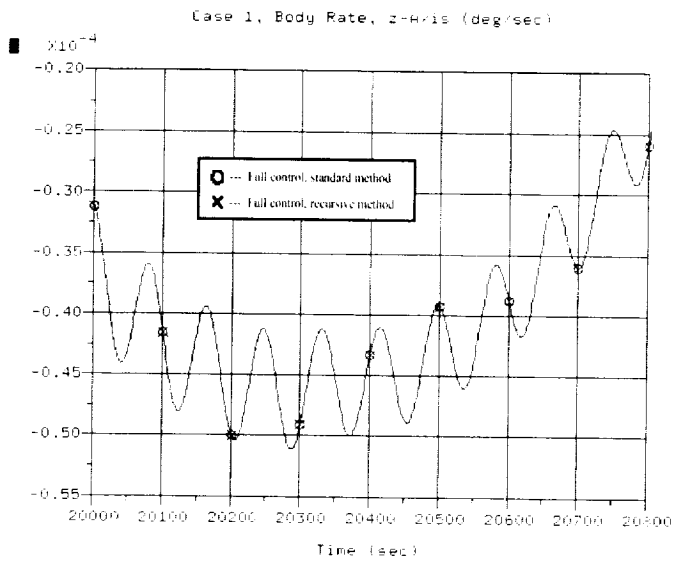

Case 1, Body Rate, y-Axis (deg/sec)
x10^-4
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
-0.02
-0.04
-0.06
59500
59600
59700
59800
59900
60000
59550
59650
59750
59850
59950
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

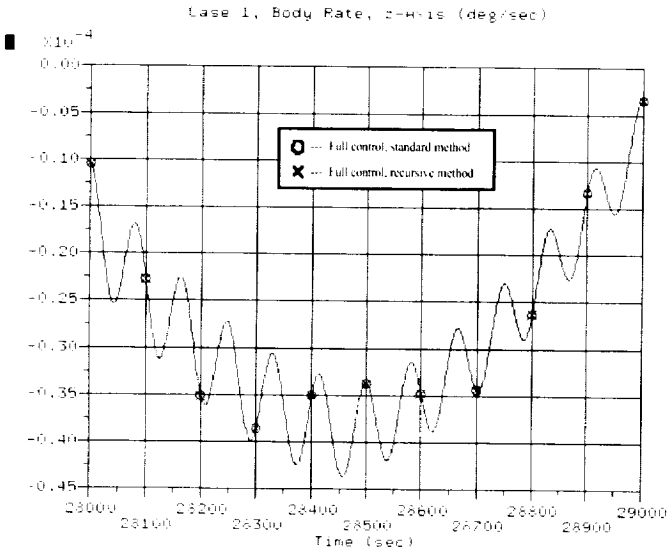

Case 1, Body Rate, z-Axis (deg/sec)
0.0003
0.0002
0.0001
0.0000
-0.0001
-0.0002
-0.0003
0
4000
8000
12000
16000
20000
2000
6000
10000
14000
18000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

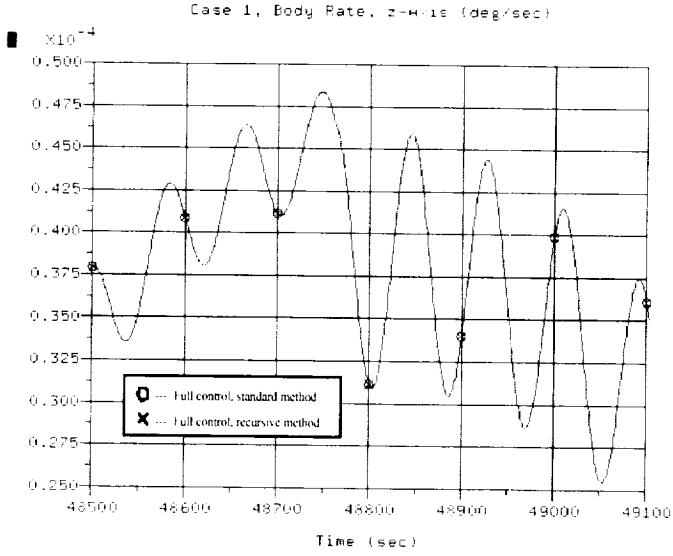

Case 1, Body Rate, z-Axis (deg/sec)
x10^-4
0.6
0.4
0.2
0.0
-0.2
-0.4
-0.6
20000
24000
28000
32000
36000
40000
22000
26000
30000
34000
38000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

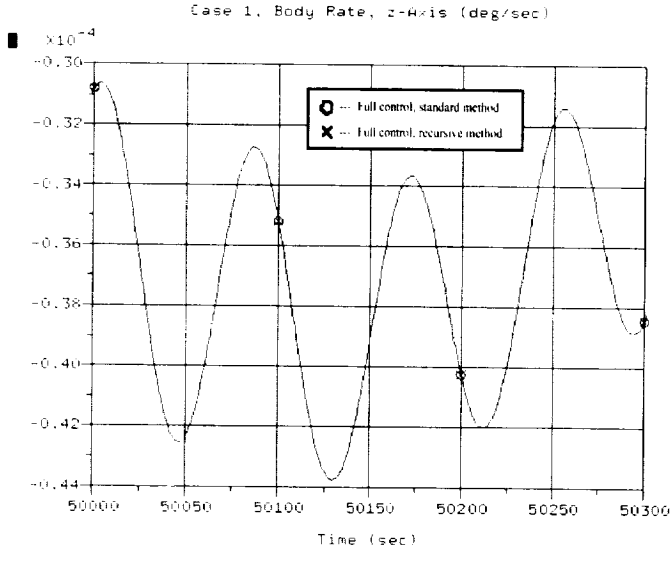

Case 1, Body Rate, z-Axis (deg/sec)
x10^-4
0.6
0.4
0.2
0.0
-0.2
-0.4
-0.6
40000
44000
48000
52000
56000
60000
42000
46000
50000
54000
58000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

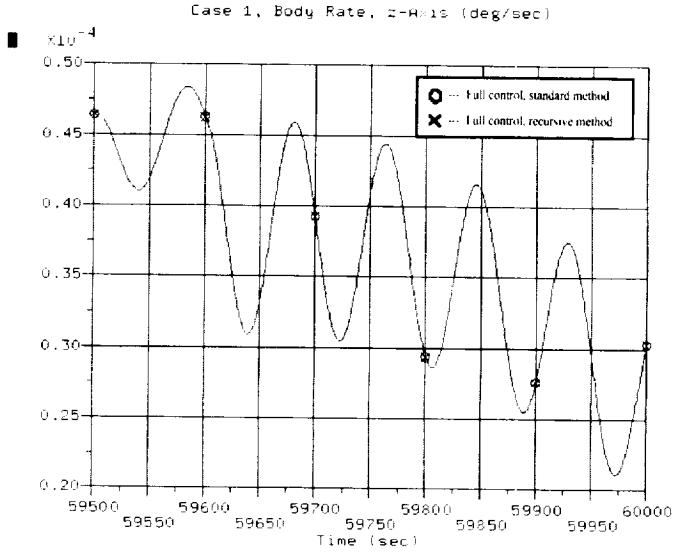

Case 1, Body Rate, z-Axis (deg/sec)

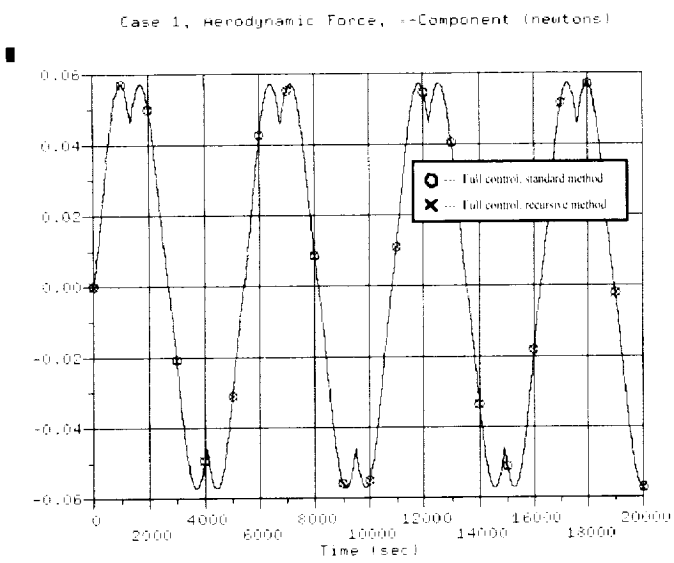

Case 1, Body Rate, z-Axis (deg/sec)

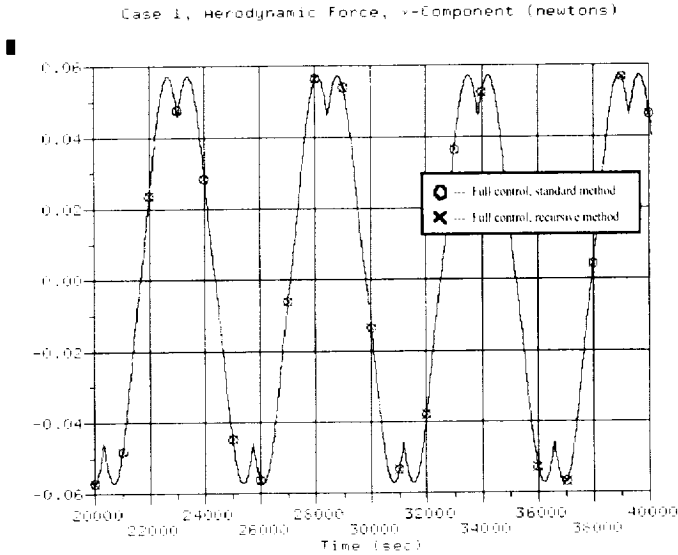

Case 1, Body Rate, z-Axis (deg/sec)

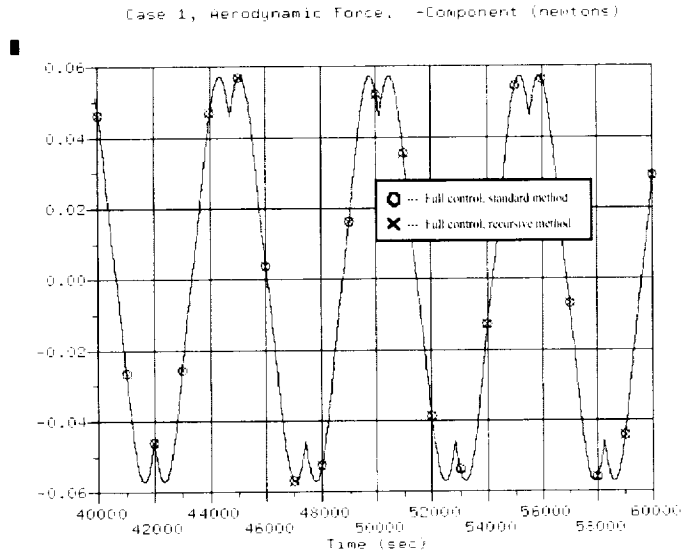

Case 1, Body Rate, z-Axis (deg/sec)

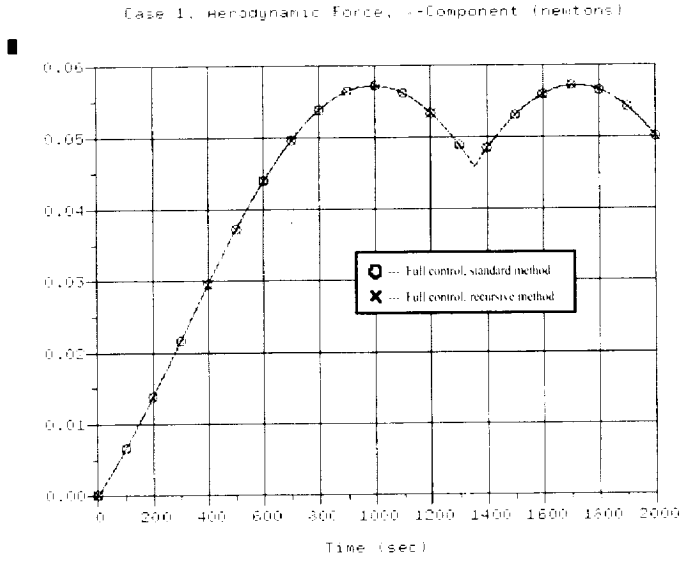

Case 1, Body Rate, z-Axis (deg/sec)

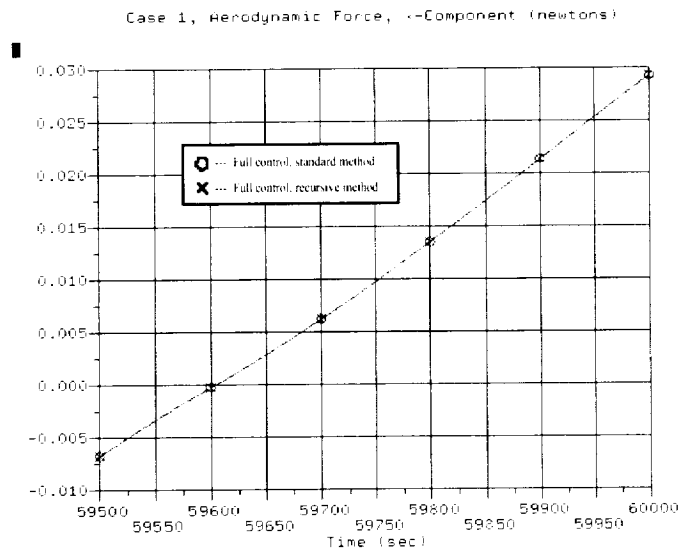

Case 1, Body Rate, z-Axis (deg/sec)

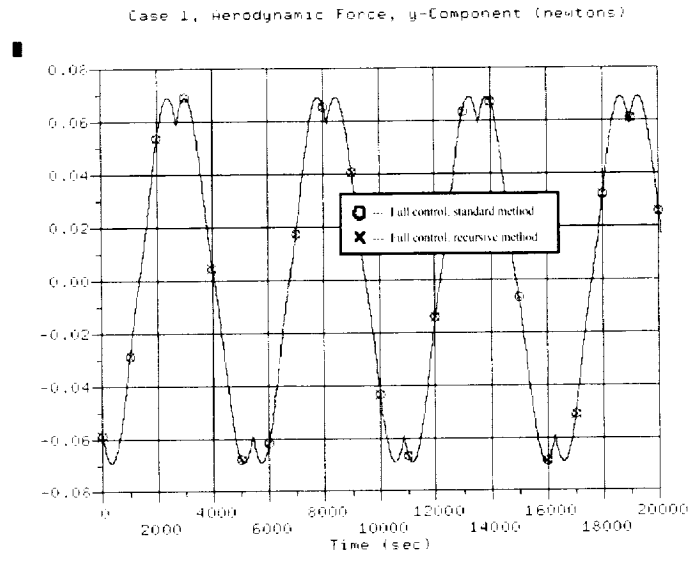

Case 1, Aerodynamic Force, x-Component (newtons)
O --- Full control, standard method
X --- Full control, recursive method
Time (sec)

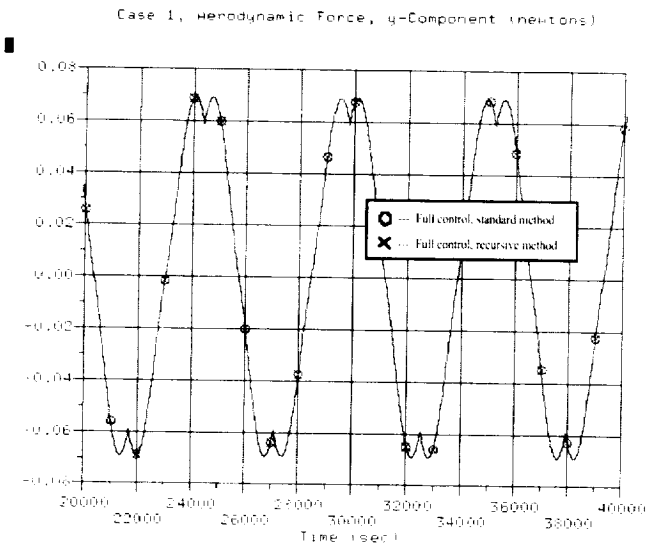

Case 1, Aerodynamic Force, y-Component (newtons)
O --- Full control, standard method
X --- Full control, recursive method
Time (sec)

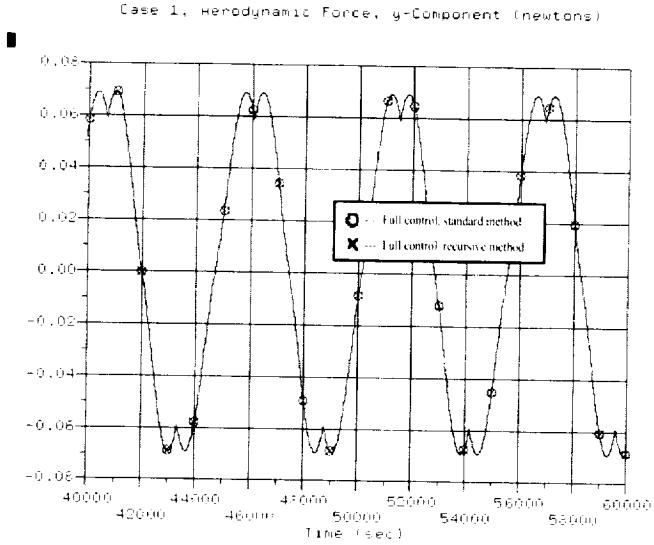

Case 1, Aerodynamic Force, z-Component (newtons)
O --- Full control, standard method
X --- Full control, recursive method
Time (sec)

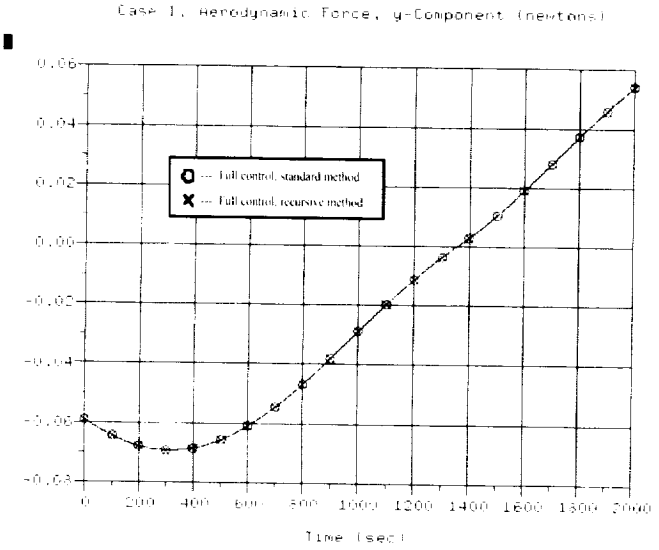

Case 1, Aerodynamic Force, x-Component (newtons)
O --- Full control, standard method
X --- Full control, recursive method
Time (sec)

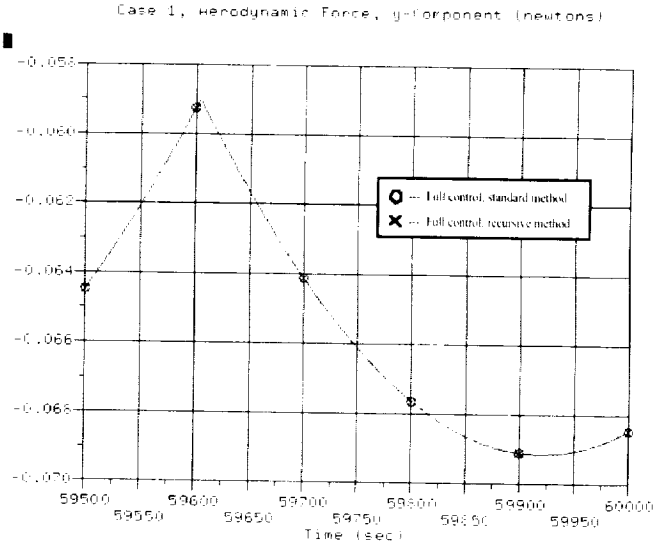

Case 1, Aerodynamic Force, x-Component (newtons)
O --- Full control, standard method
X --- Full control, recursive method
Time (sec)

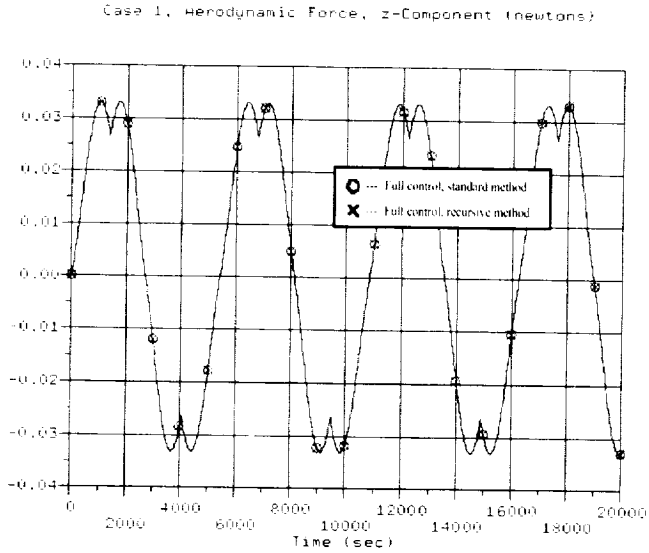

Case 1, Aerodynamic Force, y-Component (newtons)
O --- Full control, standard method
X --- Full control, recursive method
Time (sec)

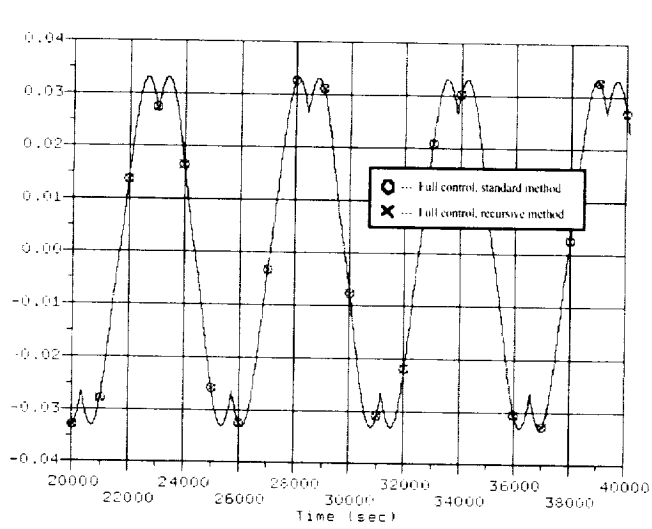

Case 1, Aerodynamic Force, y-Component (newtons)
0.08
0.06
0.04
0.02
0.00
-0.02
-0.04
-0.06
-0.08
20000
22000
24000
26000
28000
30000
32000
34000
36000
38000
40000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method


Case 1, Aerodynamic Force, y-Component (newtons)
0.08
0.06
0.04
0.02
0.00
-0.02
-0.04
-0.06
-0.08
40000
42000
44000
46000
48000
50000
52000
54000
56000
58000
60000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

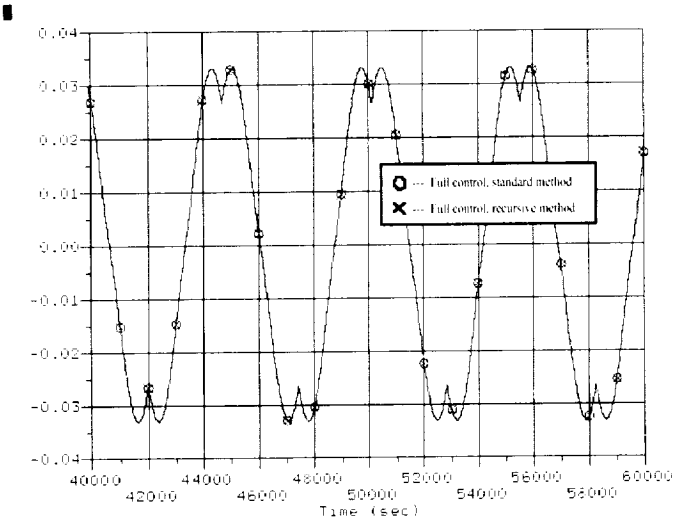

Case 1, Aerodynamic Force, y-Component (newtons)
0.06
0.04
0.02
0.00
-0.02
-0.04
-0.06
0
200
400
600
800
1000
1200
1400
1600
1800
2000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method


Case 1, Aerodynamic Force, y-Component (newtons)
-0.058
-0.060
-0.062
-0.064
-0.066
-0.068
-0.070
59500
59550
59600
59650
59700
59750
59800
59850
59900
59950
60000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

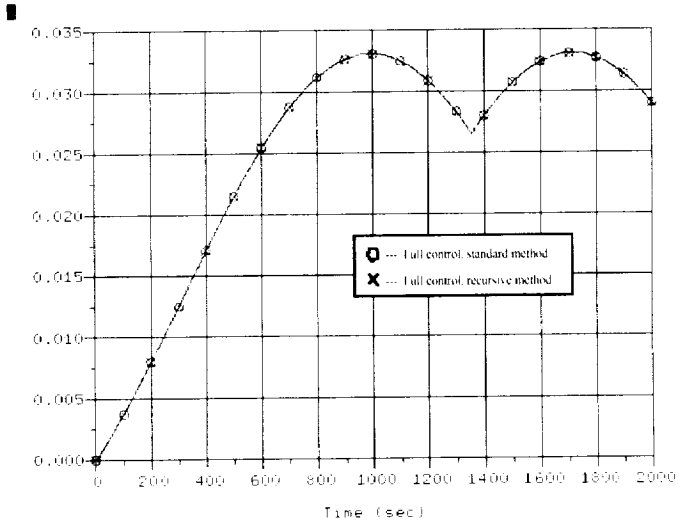

Case 1, Aerodynamic Force, z-Component (newtons)
0.04
0.03
0.02
0.01
0.00
-0.01
-0.02
-0.03
-0.04
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method


Case 1, Aerodynamic Force, z-Component (newtons)
0.04
0.03
0.02
0.01
0.00
-0.01
-0.02
-0.03
-0.04
20000
22000
24000
26000
28000
30000
32000
34000
36000
38000
40000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

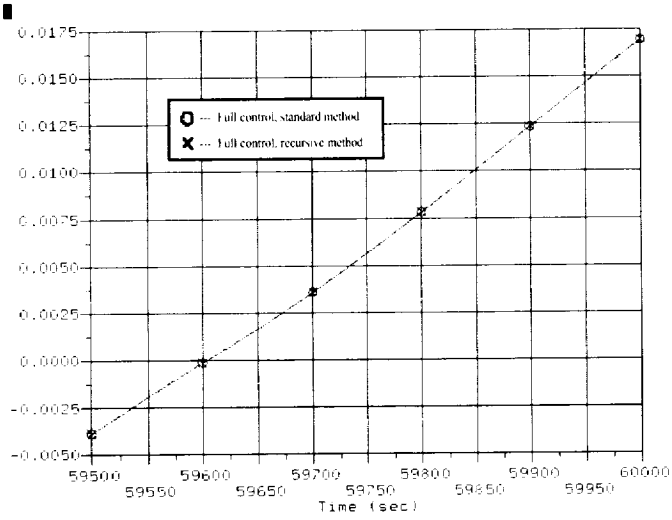
Case 1, Aerodynamic Force, z-Component (newtons)



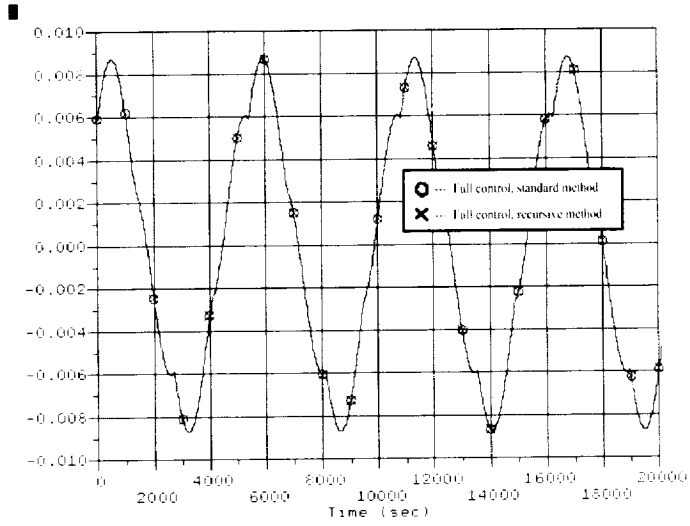
Case 1, Aerodynamic Force, z-Component (newtons)



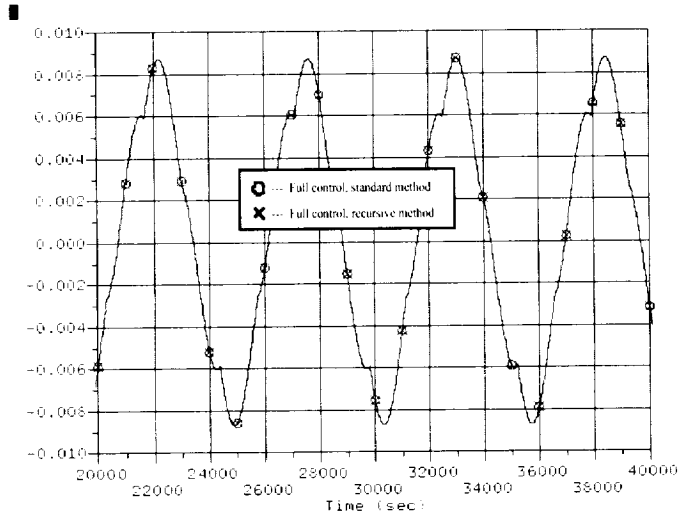
Case 1, Aerodynamic Force, z-Component (newtons)



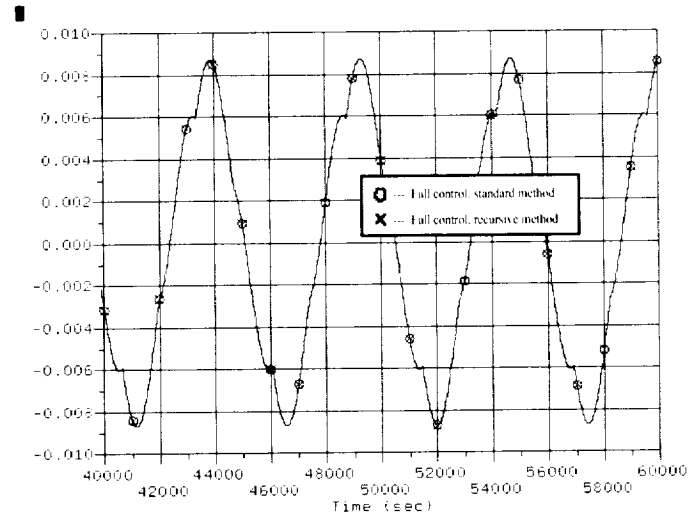
Case 1, Aerodynamic Torque, x-Component (newton-meters)

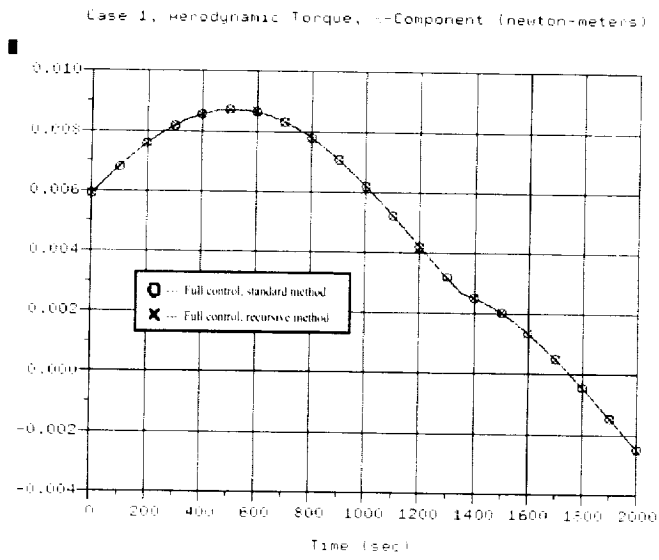


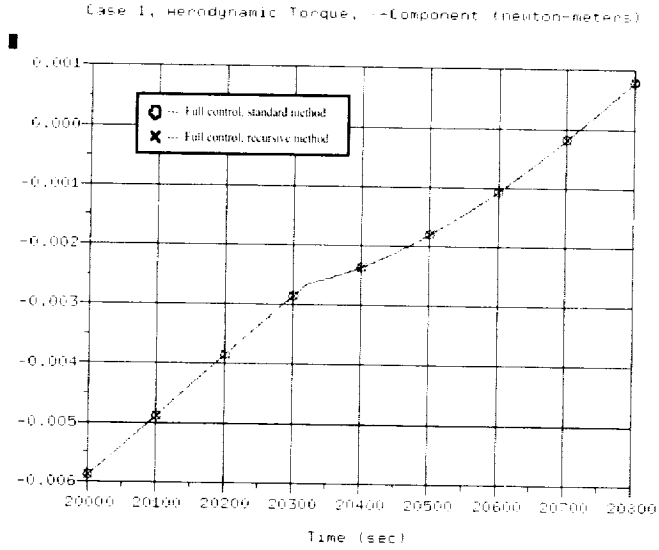
Case 1, Aerodynamic Torque, x-Component (newton-meters)

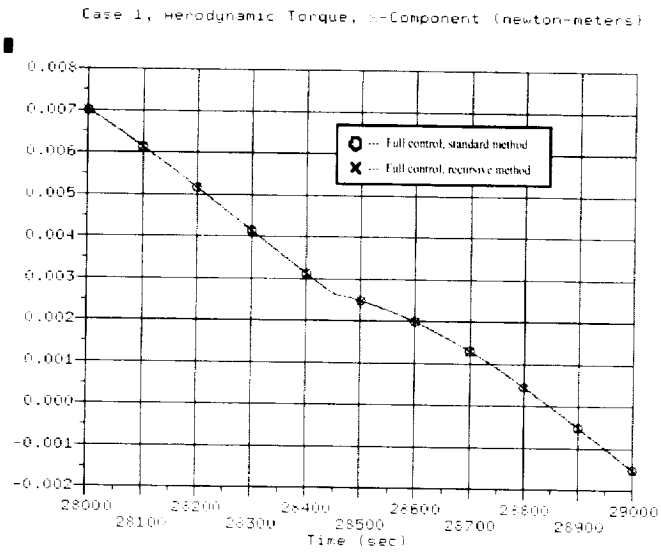


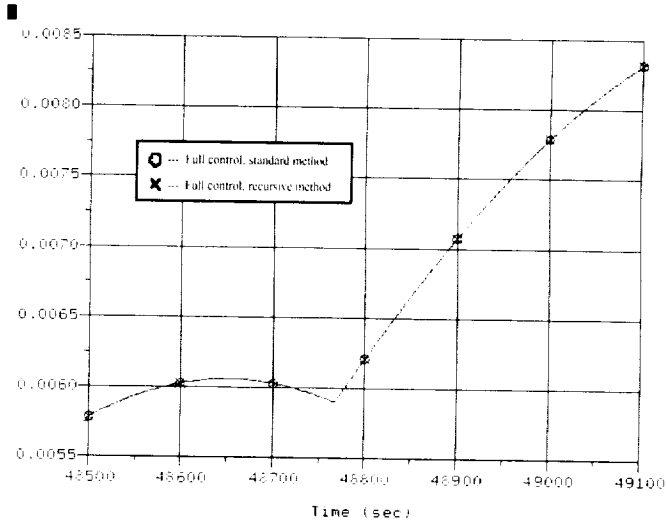
Case 1, Aerodynamic Torque, x-Component (newton-meters)

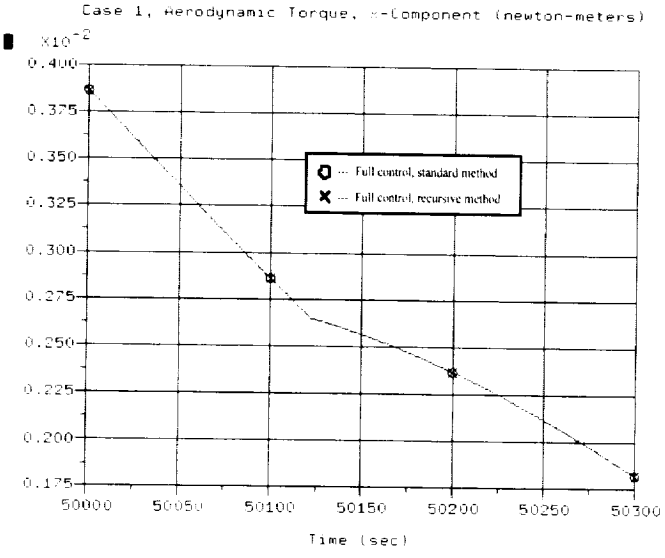


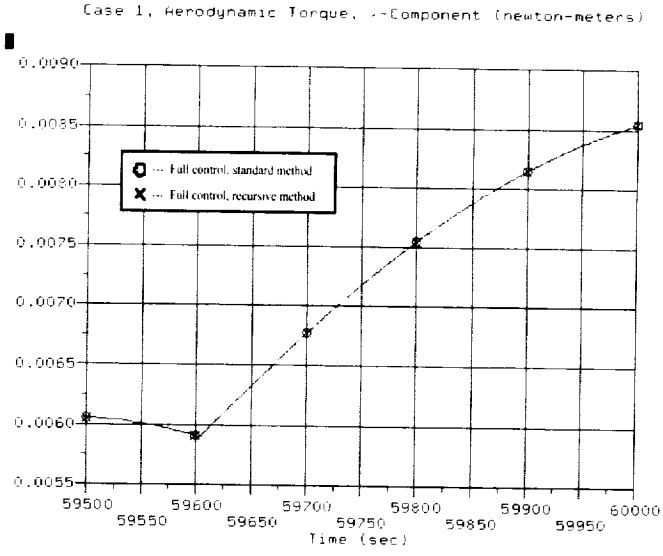

Case 1, Aerodynamic Torque, x-Component (newton-meters)
0.010
0.008
0.006
0.004
0.002
0.000
-0.002
-0.004
0
200
400
600
800
1000
1200
1400
1600
1800
2000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

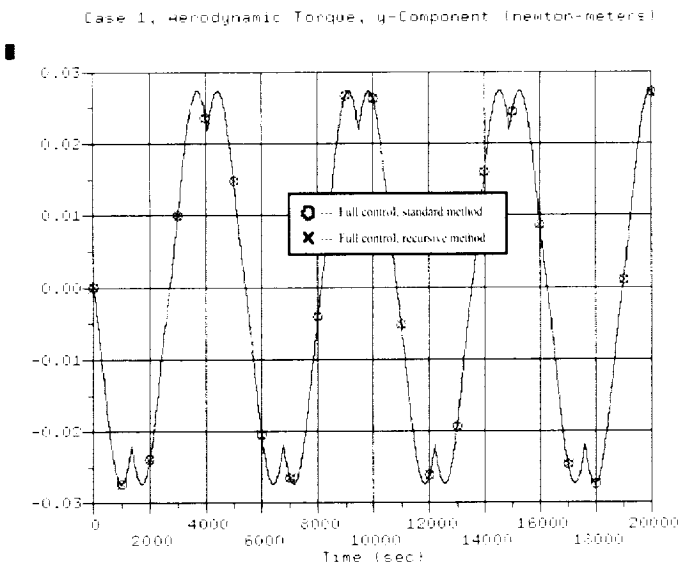

Case 1, Aerodynamic Torque, x-Component (newton-meters)
0.001
0.000
-0.001
-0.002
-0.003
-0.004
-0.005
-0.006
20000
20100
20200
20300
20400
20500
20600
20700
20800
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

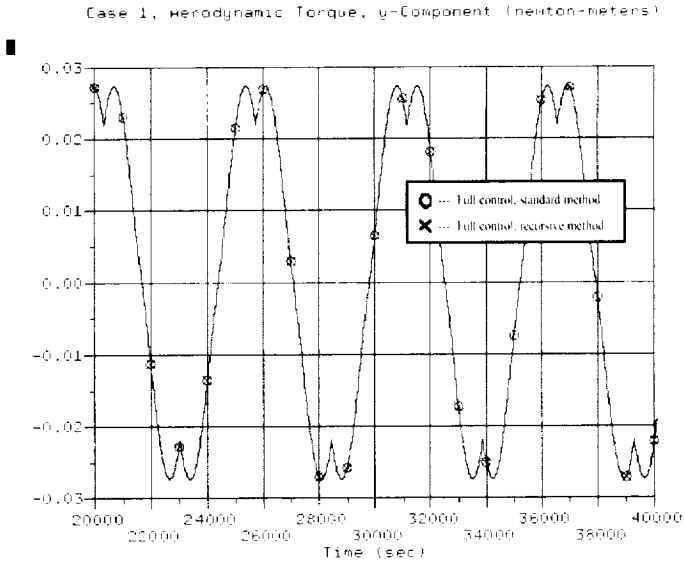

Case 1, Aerodynamic Torque, x-Component (newton-meters)
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
-0.001
-0.002
28000
28100
28200
28300
28400
28500
28600
28700
28800
28900
29000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

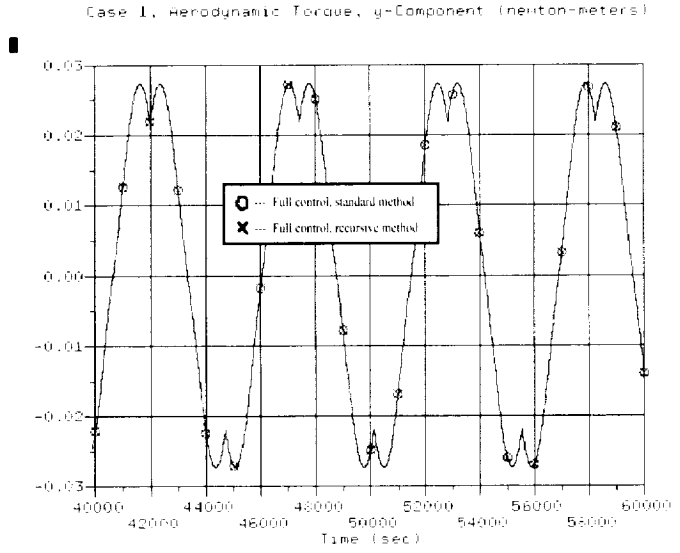

Case 1, Aerodynamic Torque, x-Component (newton-meters)
0.0085
0.0080
0.0075
0.0070
0.0065
0.0060
0.0055
48500
48600
48700
48800
48900
49000
49100
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

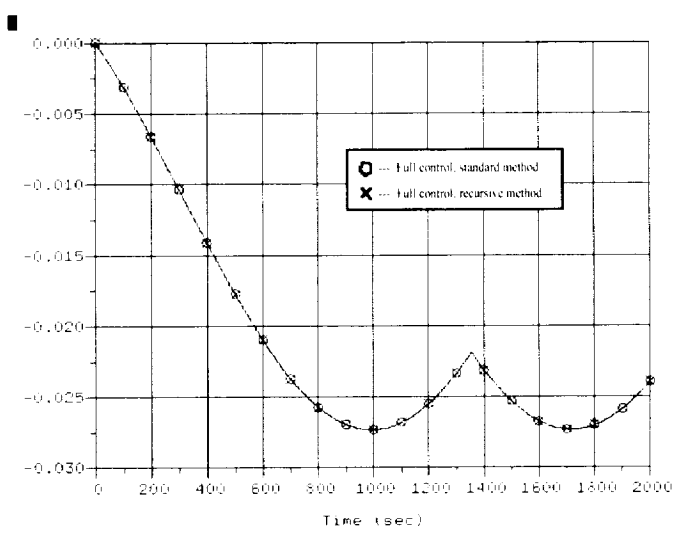

Case 1, Aerodynamic Torque, x-Component (newton-meters)
X10^-2
0.400
0.375
0.350
0.325
0.300
0.275
0.250
0.225
0.200
0.175
50000
50050
50100
50150
50200
50250
50300
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

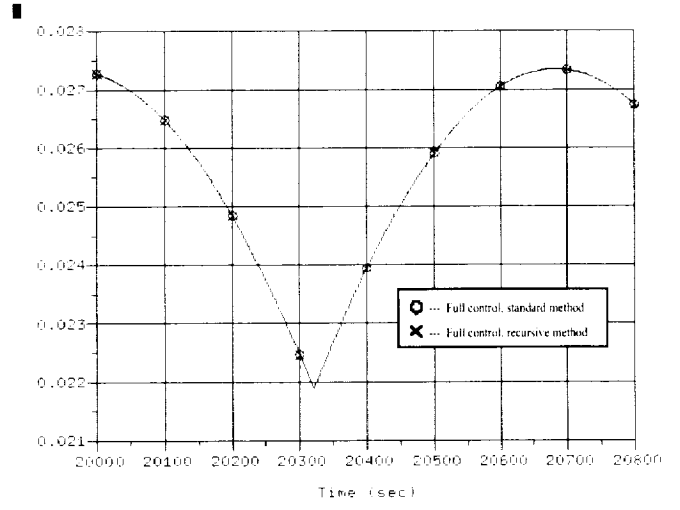

Case 1, Aerodynamic Torque, x-Component (newton-meters)
0.0090
0.0085
0.0080
0.0075
0.0070
0.0065
0.0060
0.0055
59500
59550
59600
59650
59700
59750
59800
59850
59900
59950
60000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

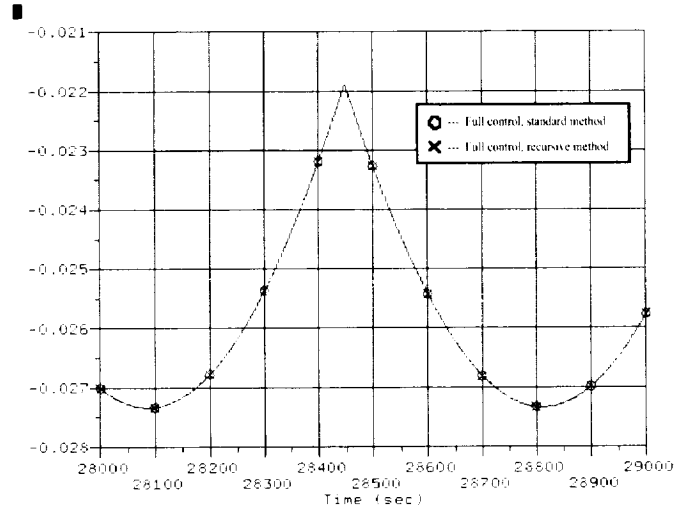

Case 1, Aerodynamic Torque, y-Component (newton-meters)
0.03
0.02
0.01
0.00
-0.01
-0.02
-0.03
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

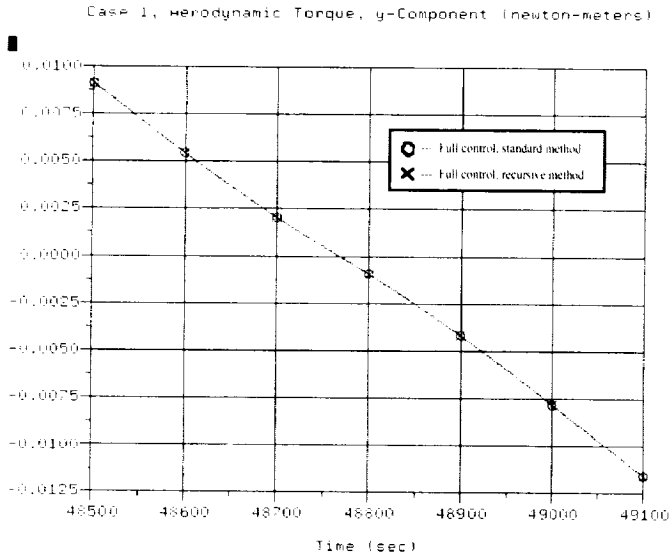

Case 1, Aerodynamic Torque, y-Component (newton-meters)
0.03
0.02
0.01
0.00
-0.01
-0.02
-0.03
20000
22000
24000
26000
28000
30000
32000
34000
36000
38000
40000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

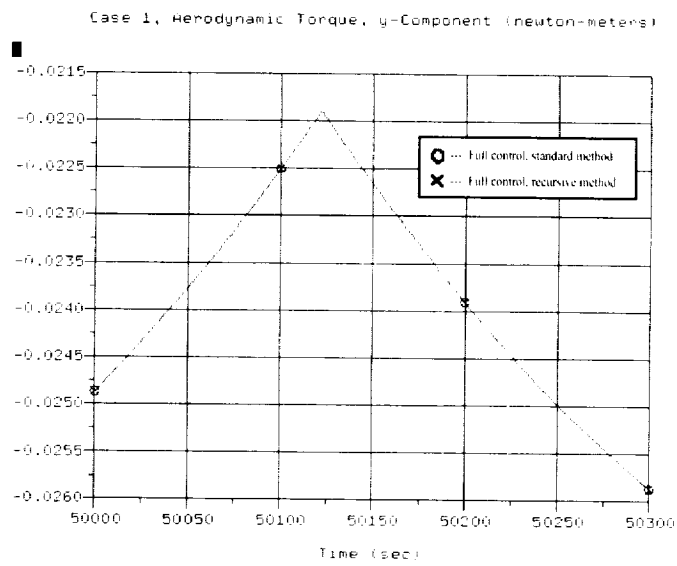

Case 1, Aerodynamic Torque, y-Component (newton-meters)
0.03
0.02
0.01
0.00
-0.01
-0.02
-0.03
40000
42000
44000
46000
48000
50000
52000
54000
56000
58000
60000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

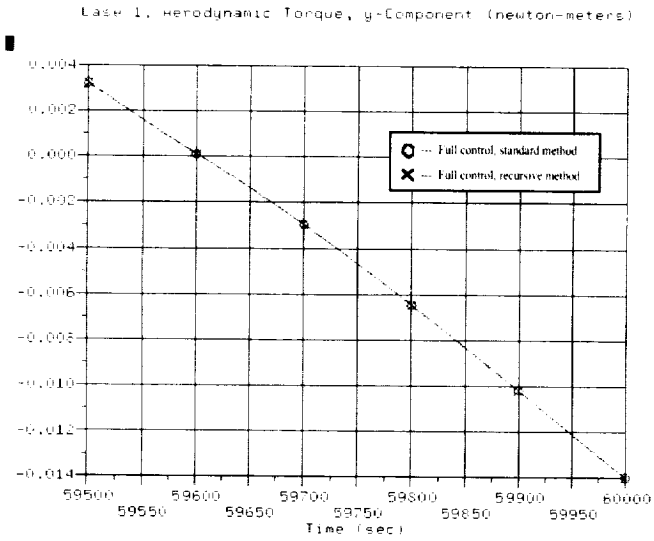

Case 1, Aerodynamic Torque, y-Component (newton-meters)
0.000
-0.005
-0.010
-0.015
-0.020
-0.025
-0.030
0
200
400
600
800
1000
1200
1400
1600
1800
2000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

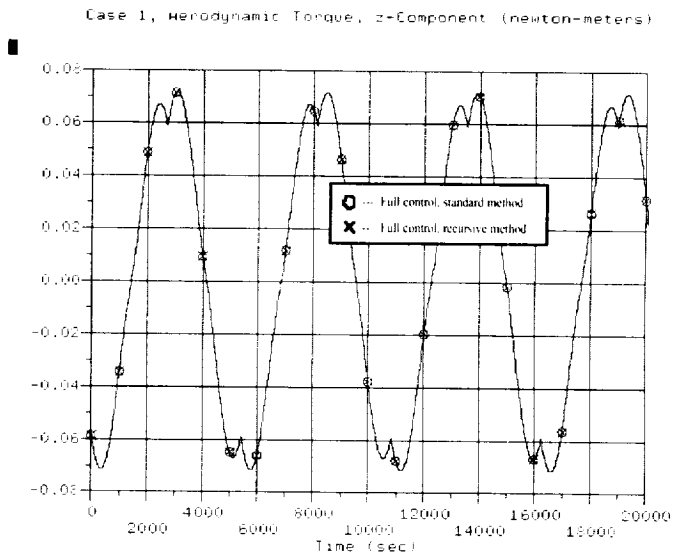

Case 1, Aerodynamic Torque, y-Component (newton-meters)
0.028
0.027
0.026
0.025
0.024
0.023
0.022
0.021
20000
20100
20200
20300
20400
20500
20600
20700
20800
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

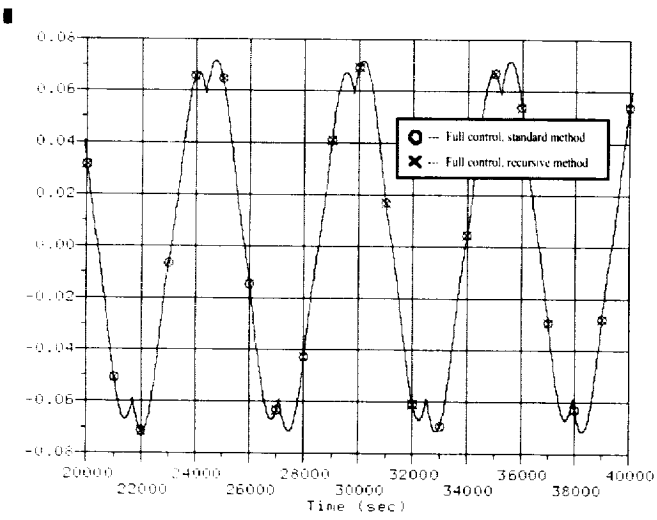

Case 1, Aerodynamic Torque, y-Component (newton-meters)
-0.021
-0.022
-0.023
-0.024
-0.025
-0.026
-0.027
-0.028
28000
28100
28200
28300
28400
28500
28600
28700
28800
28900
29000
Time (sec)
O --- Full control, standard method
X --- Full control, recursive method

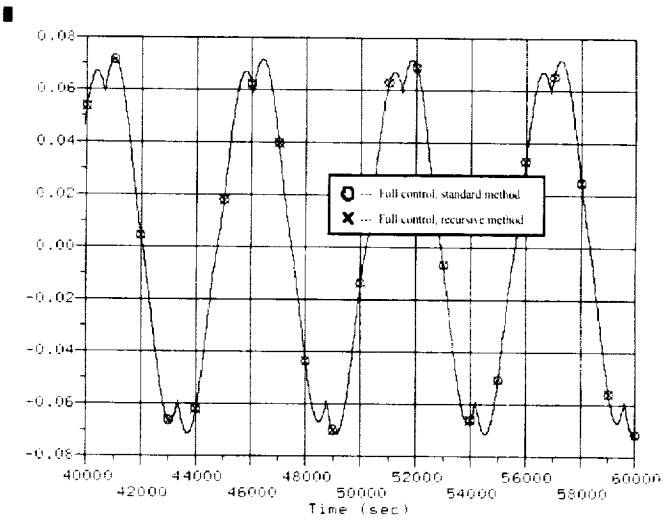

Case 1, Aerodynamic Torque, y-Component (newton-meters)
Full control, standard method
Full control, recursive method
Time (sec)

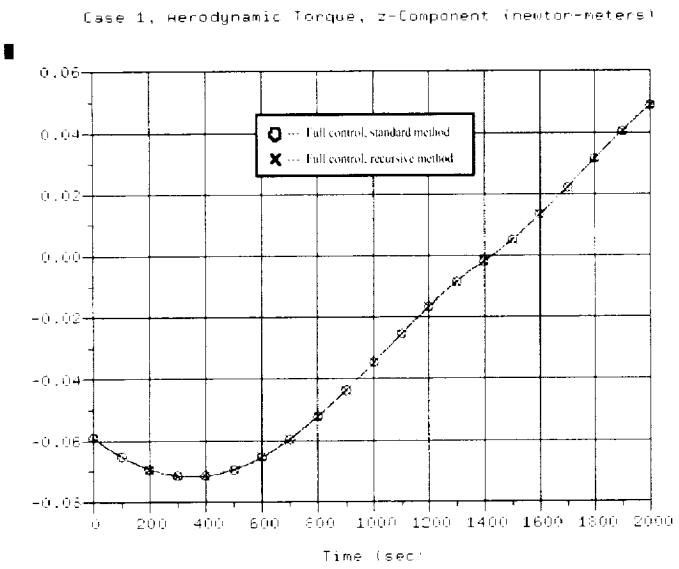

Case 1, Aerodynamic Torque, y-Component (newton-meters)
Full control, standard method
Full control, recursive method
Time (sec)

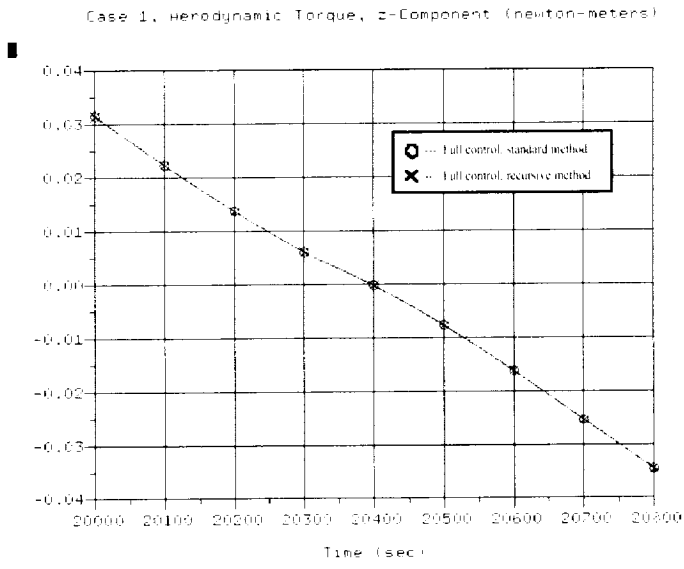

Case 1, Aerodynamic Torque, y-Component (newton-meters)
Full control, standard method
Full control, recursive method
Time (sec)

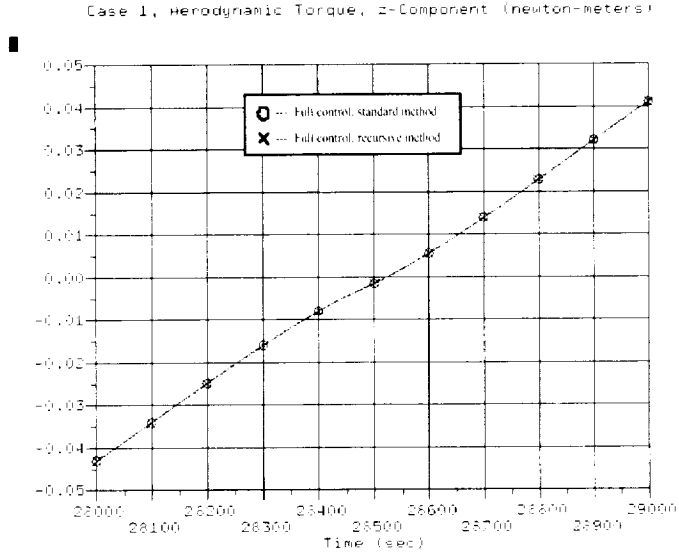

Case 1, Aerodynamic Torque, z-Component (newton-meters)
Full control, standard method
Full control, recursive method
Time (sec)

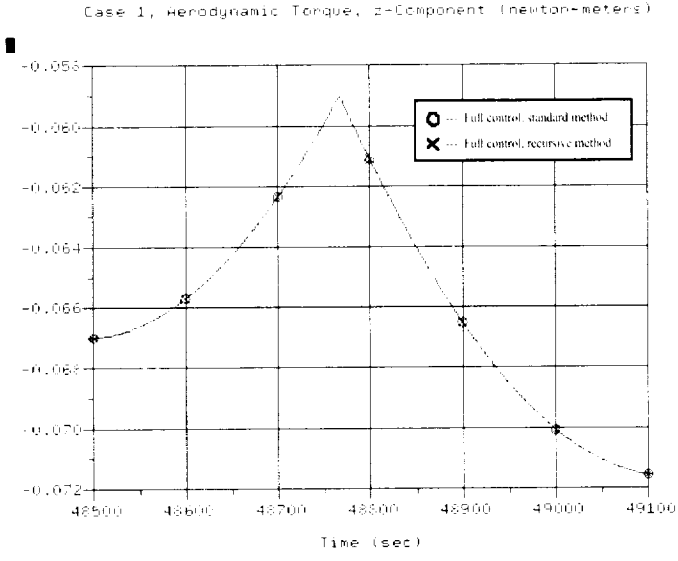

Case 1, Aerodynamic Torque, z-Component (newton-meters)
Full control, standard method
Full control, recursive method
Time (sec)

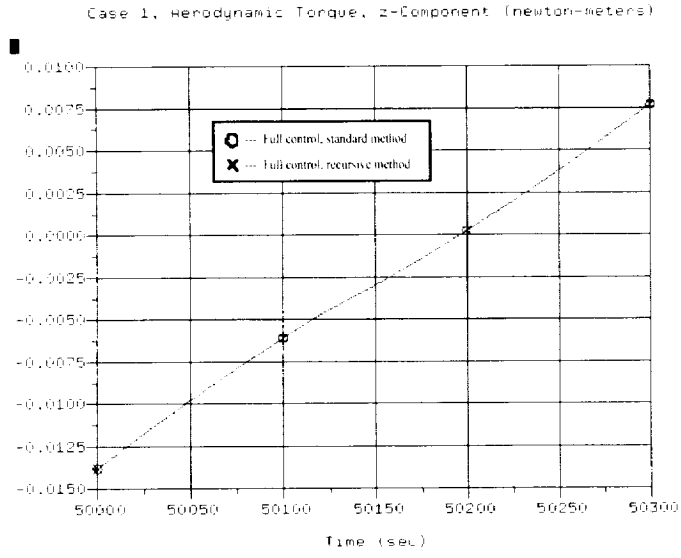

Case 1, Aerodynamic Torque, z-Component (newton-meters)
Full control, standard method
Full control, recursive method
Time (sec)

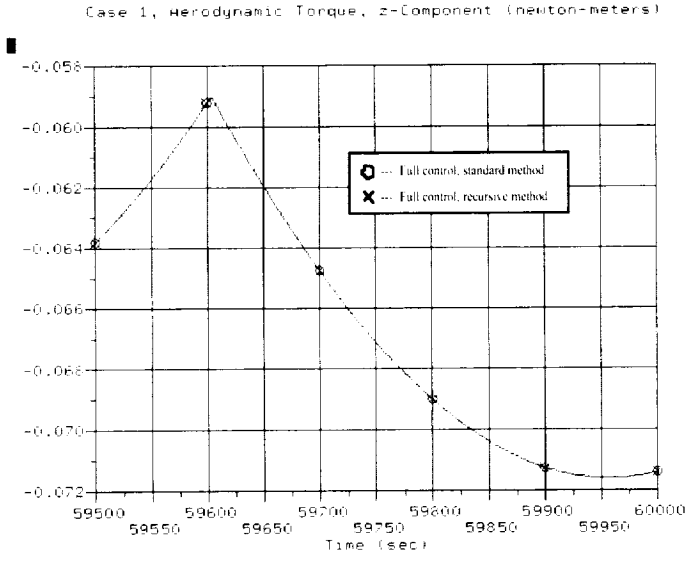

Case 1, Aerodynamic Torque, z-Component (newton-meters)


Case 1, Aerodynamic Torque, z-Component (newton-meters)

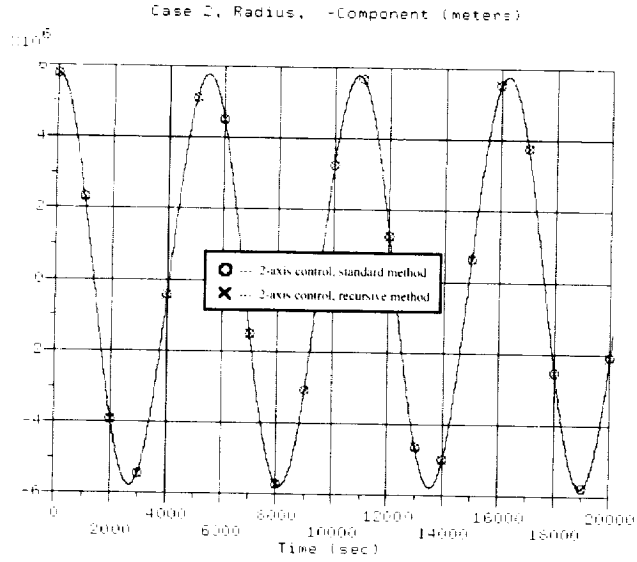

Case 1, Aerodynamic Torque, z-Component (newton-meters)

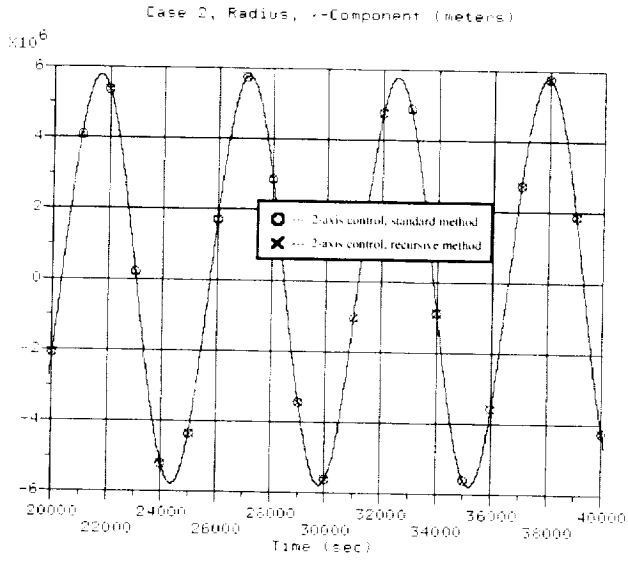

Case 1, Aerodynamic Torque, z-Component (newton-meters)

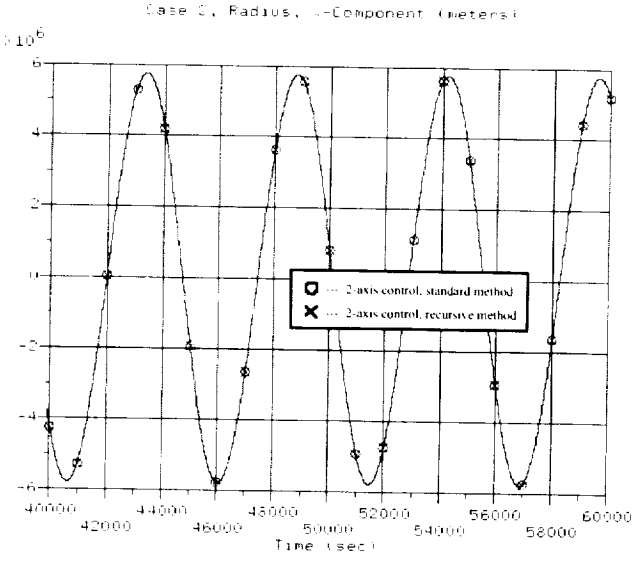

Case 1, Aerodynamic Torque, z-Component (newton-meters)

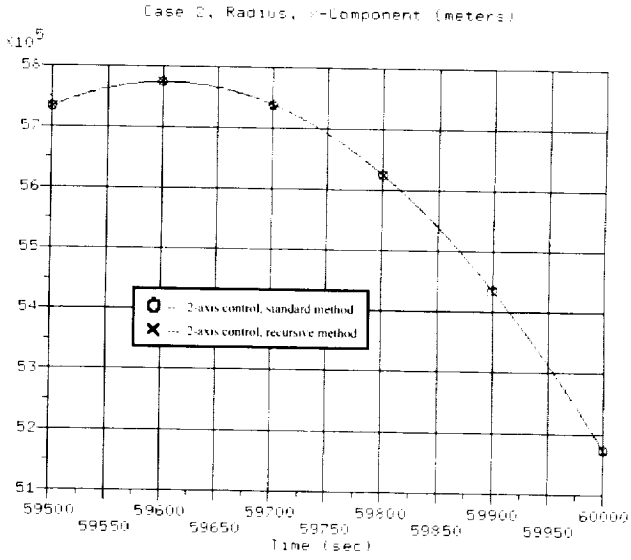

Case 1, Aerodynamic Torque, z-Component (newton-meters)

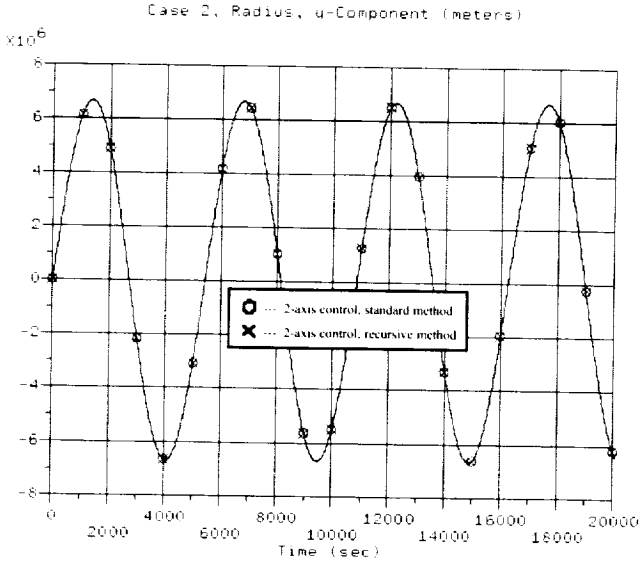
A.2 Response Plots for Case 2 (Two-Axis Attitude Control)

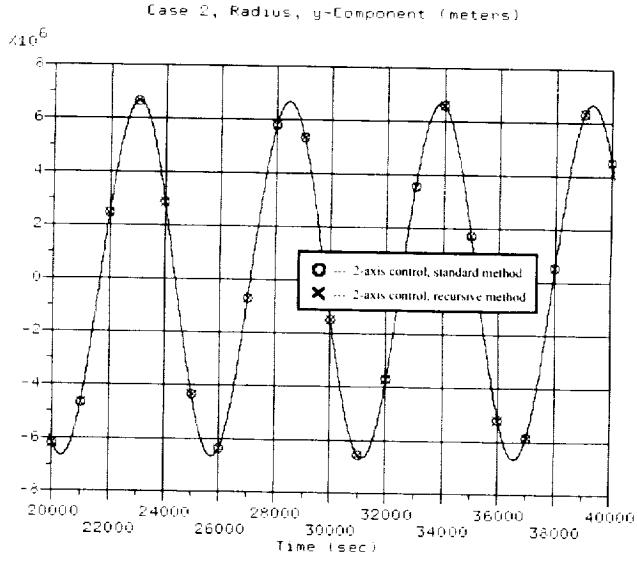

Case 2, Radius, -Component (meters)
x10^6
O ... 2-axis control, standard method
X ... 2-axis control, recursive method
Time (sec)


Case 2, Radius, -Component (meters)
x10^6
O ... 2-axis control, standard method
X ... 2-axis control, recursive method
Time (sec)

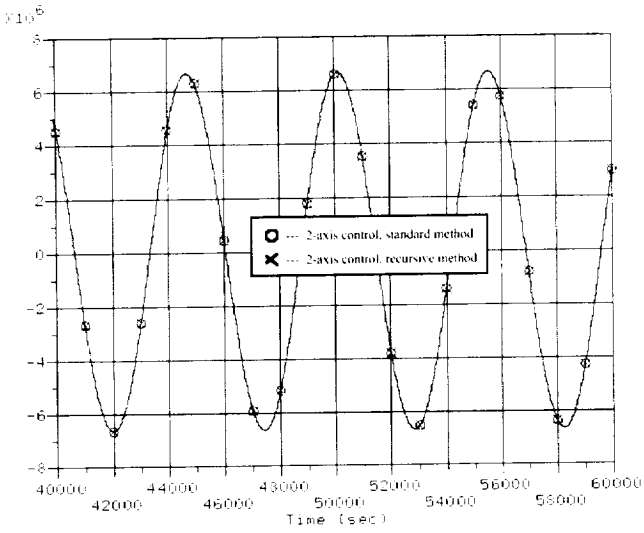

Case 2, Radius, -Component (meters)
x10^6
O ... 2-axis control, standard method
X ... 2-axis control, recursive method
Time (sec)


Case 2, Radius, -Component (meters)
x10^5
O ... 2-axis control, standard method
X ... 2-axis control, recursive method
Time (sec)

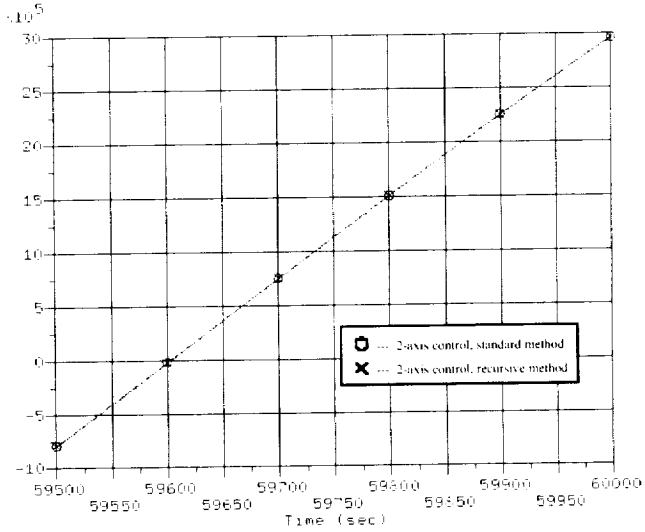

Case 2, Radius, u-Component (meters)
x10^6
O ... 2-axis control, standard method
X ... 2-axis control, recursive method
Time (sec)


Case 2, Radius, y-Component (meters)
x10^6
O ... 2-axis control, standard method
X ... 2-axis control, recursive method
Time (sec)

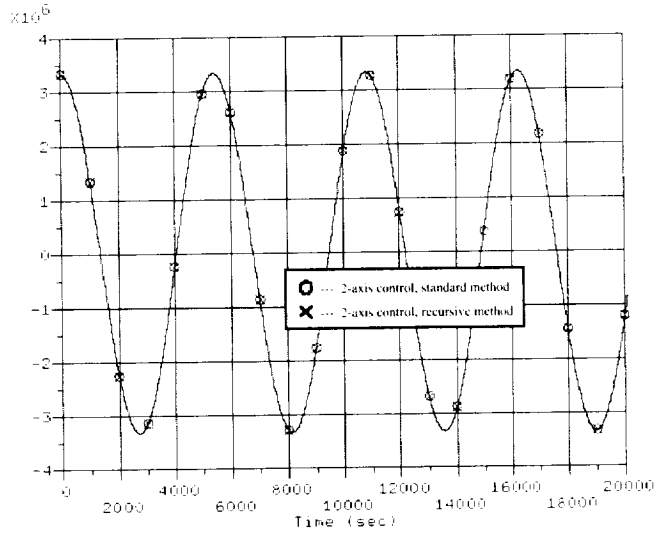
Case 2, Radius, y-Component (meters)



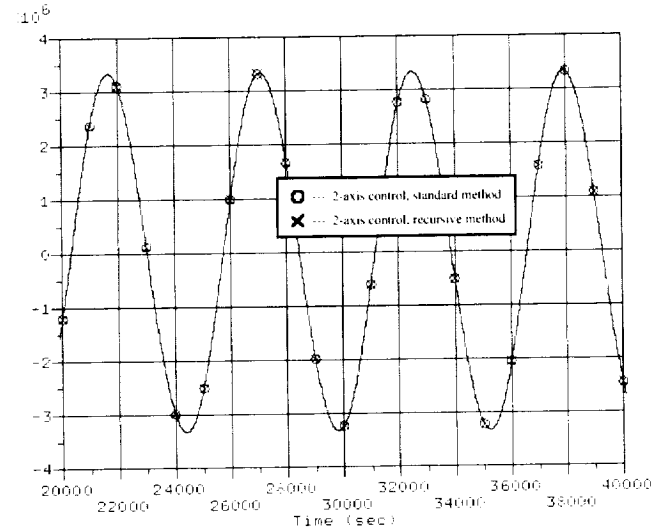
Case 2, Radius, y-Component (meters)



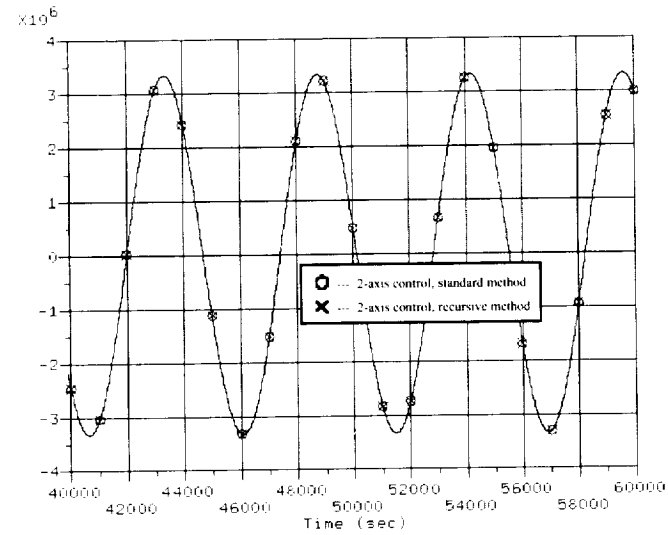
Case 2, Radius, z-Component (meters)



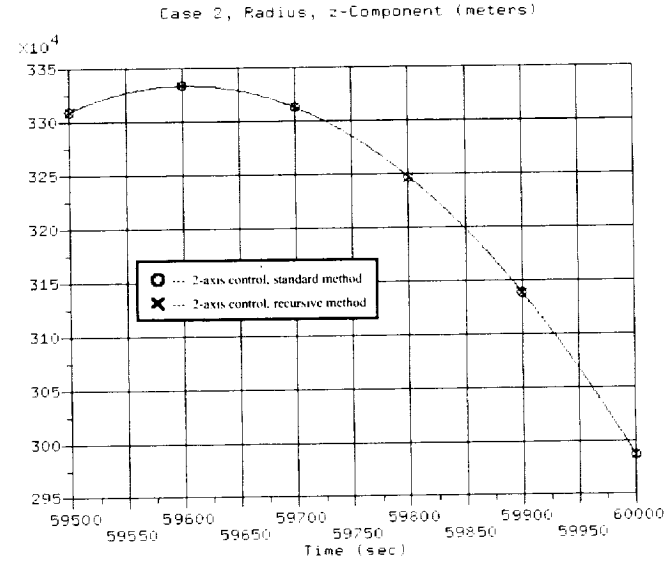
Case 2, Radius, z-Component (meters)



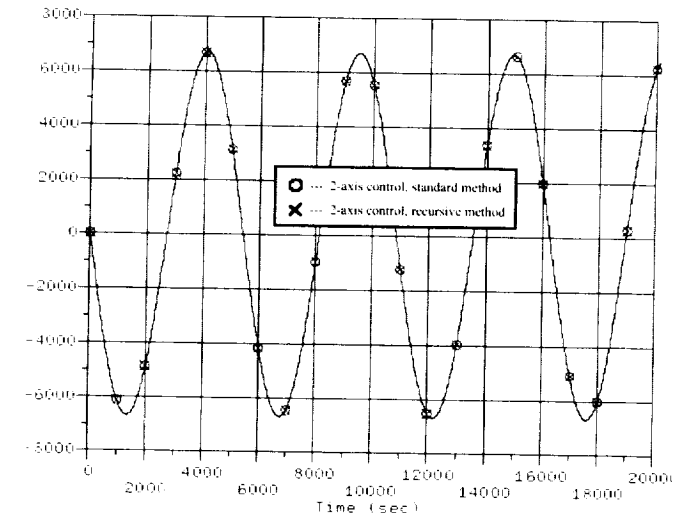
Case 2, Radius, z-Component (meters)



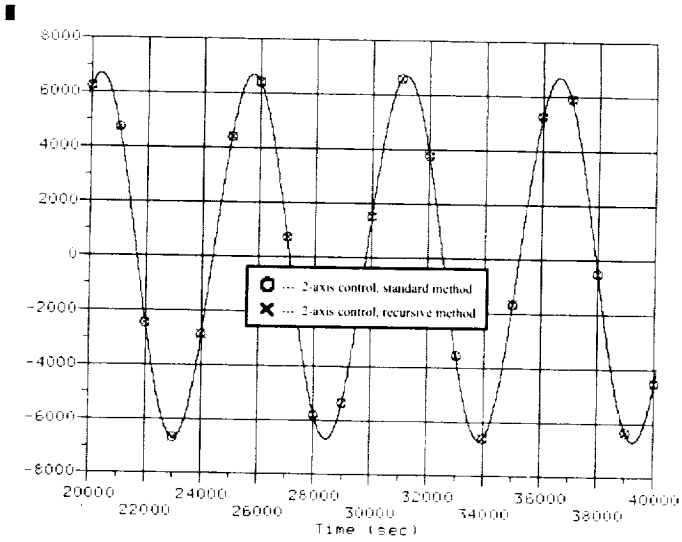
Case 2, Radius, z-Component (meters)



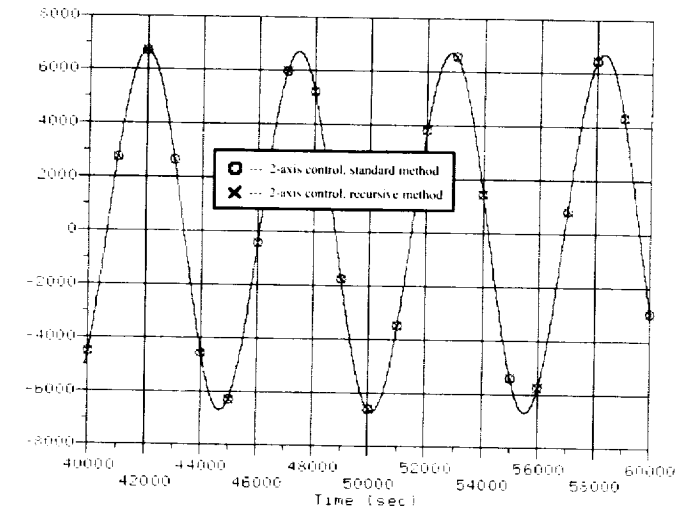
Case 2, Velocity, x-Component (meters/sec)



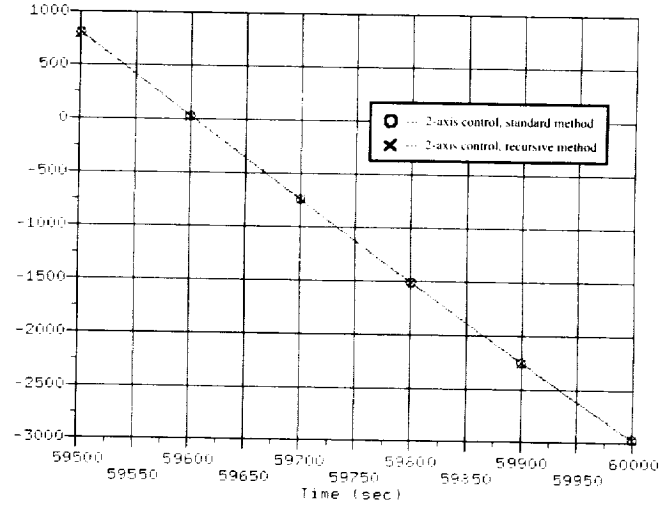
Case 2, Velocity, x-Component (meters/sec)



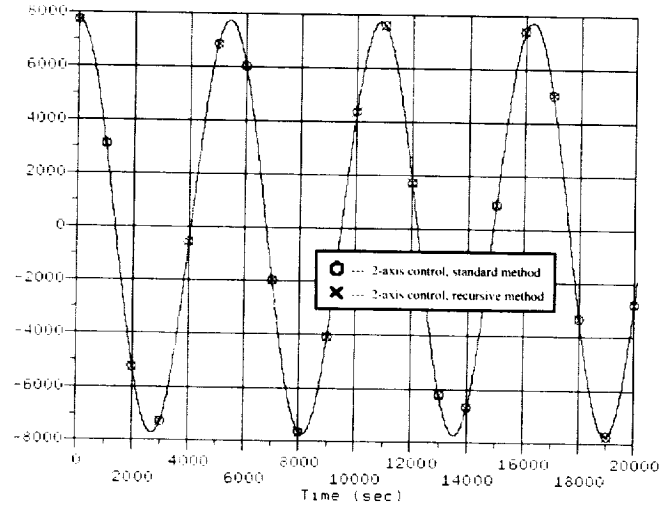
Case 2, Velocity, x-Component (meters/sec)



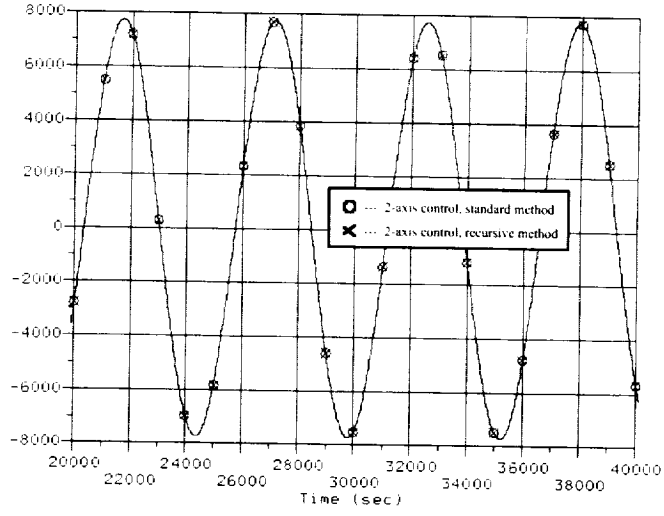
Case 2, Velocity, x-Component (meters/sec)

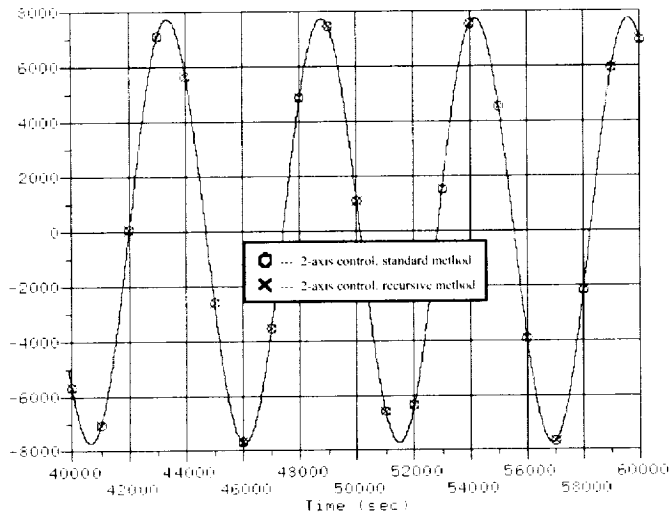


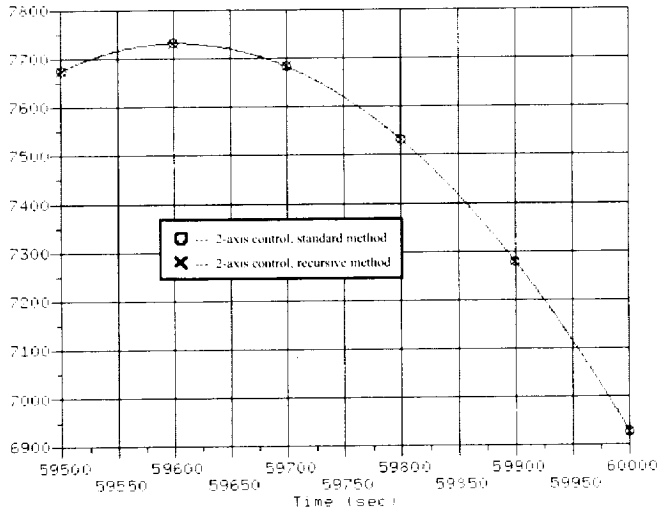
Case 2, Velocity, y-Component (meters/sec)

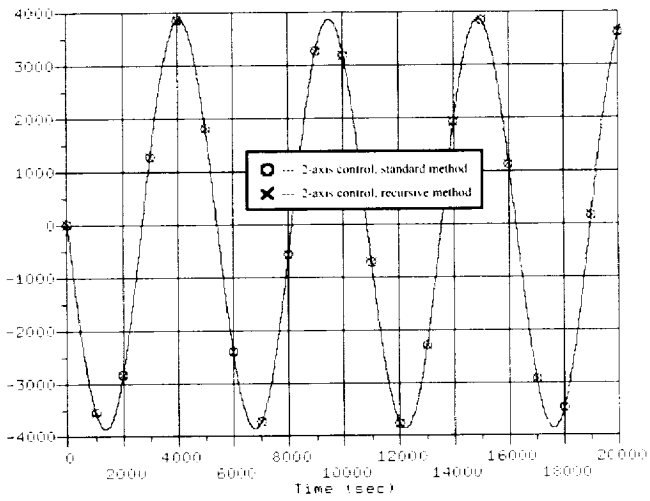


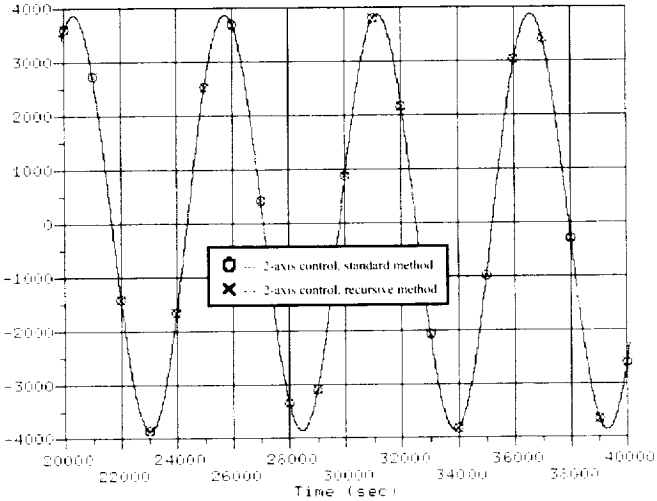
Case 2, Velocity, y-Component (meters/sec)

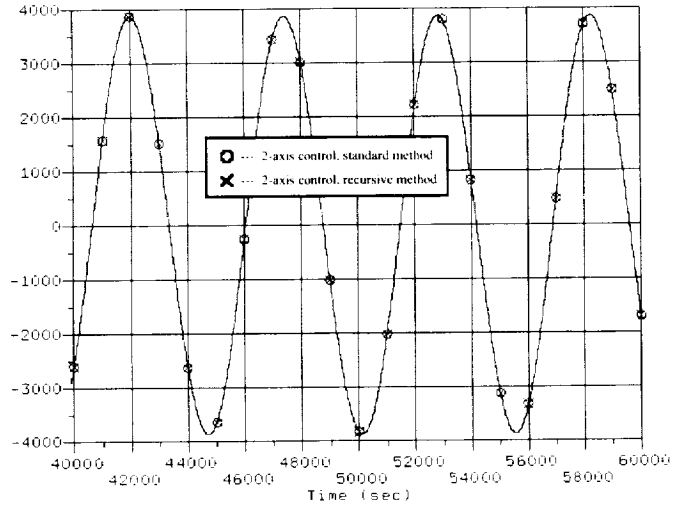


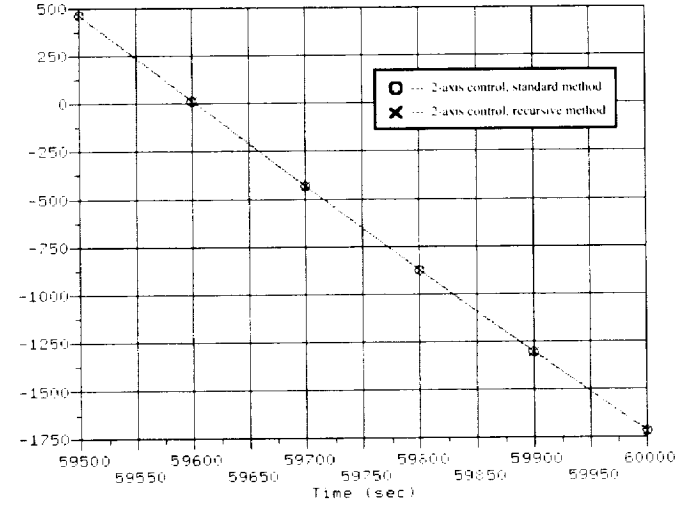

Case 2, Velocity, y-Component (meters/sec)

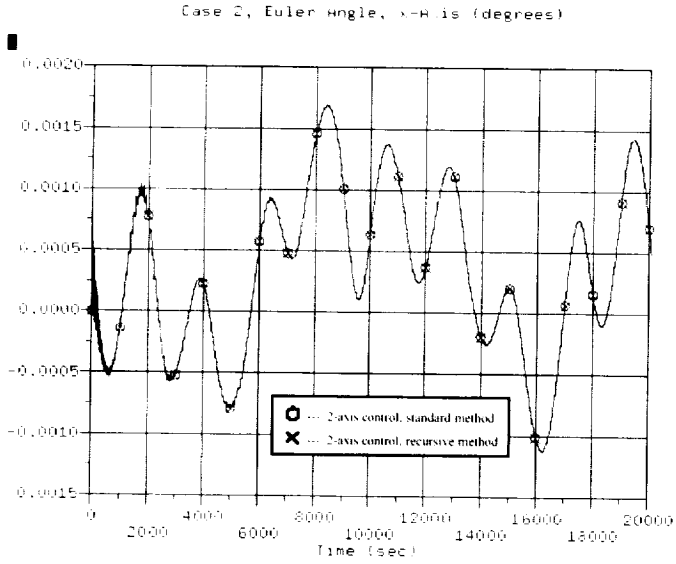

Case 2, Velocity, y-Component (meters/sec)

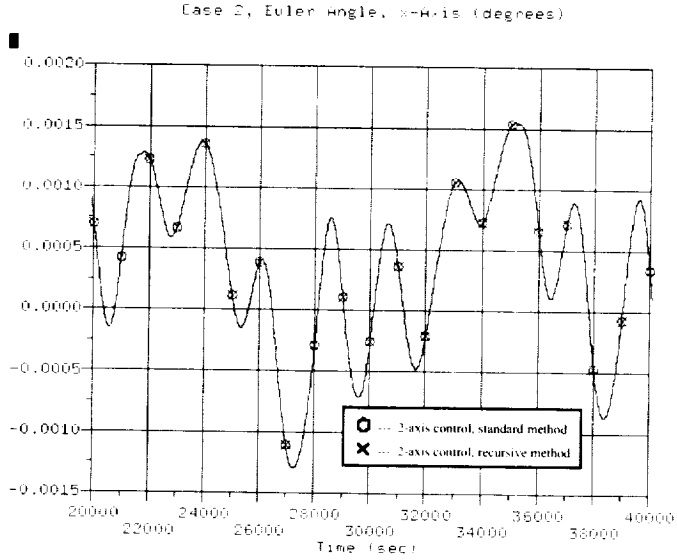

Case 2, Velocity, z-Component (meters/sec)

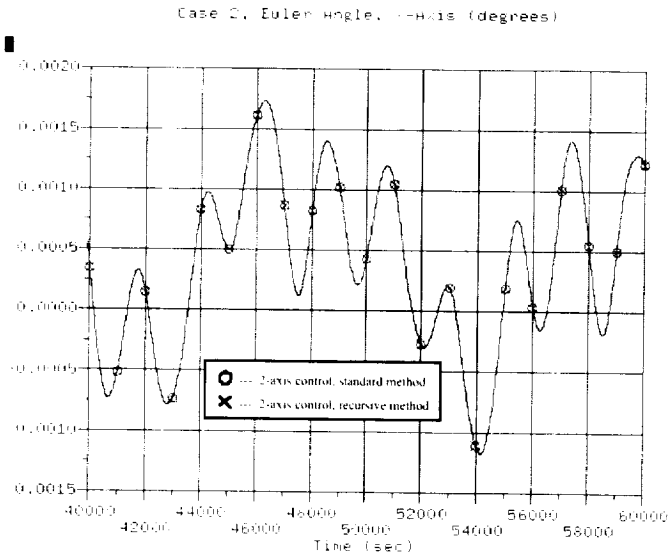

Case 2, Velocity, z-Component (meters/sec)

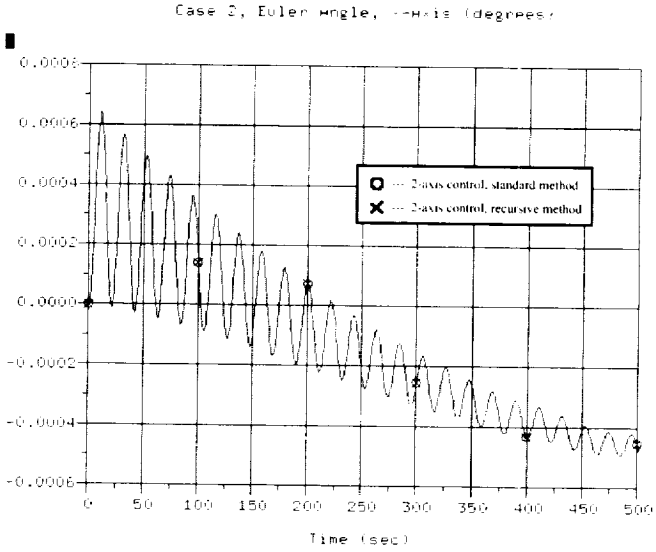

Case 2, Velocity, z-Component (meters/sec)

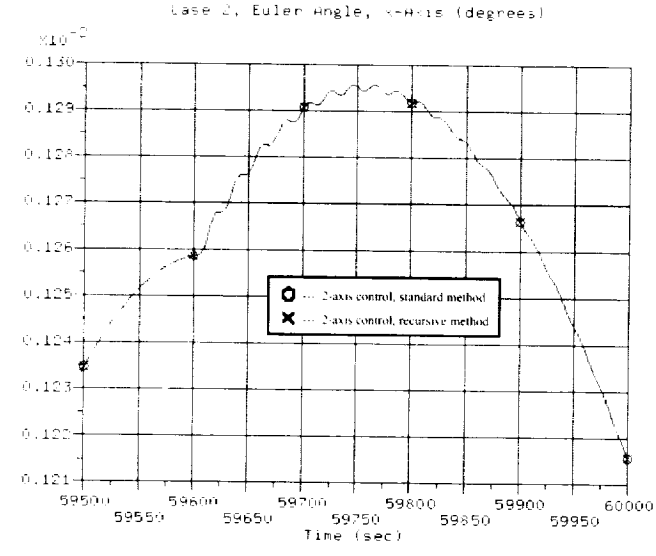

Case 2, Velocity, z-Component (meters/sec)

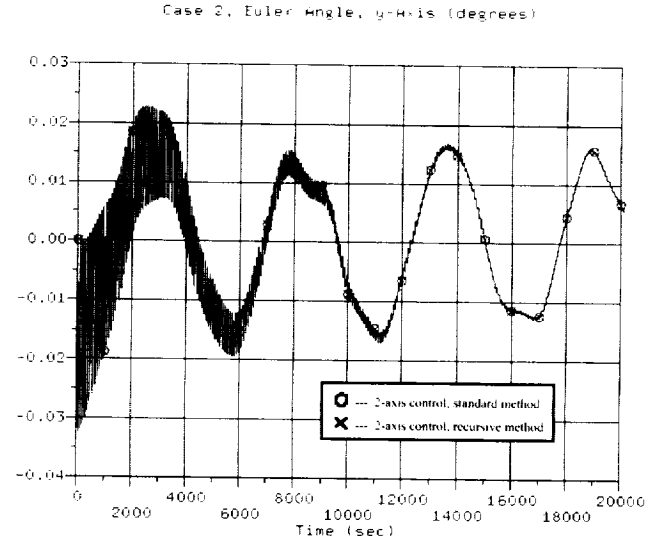

Case 2, Euler Angle, x-Axis (degrees)
0.0020
0.0015
0.0010
0.0005
0.0000
-0.0005
-0.0010
-0.0015
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Time (sec)
--- 2-axis control, standard method
--- 2-axis control, recursive method


Case 2, Euler Angle, x-Axis (degrees)
0.0020
0.0015
0.0010
0.0005
0.0000
-0.0005
-0.0010
-0.0015
20000
22000
24000
26000
28000
30000
32000
34000
36000
38000
40000
Time (sec)
--- 2-axis control, standard method
--- 2-axis control, recursive method

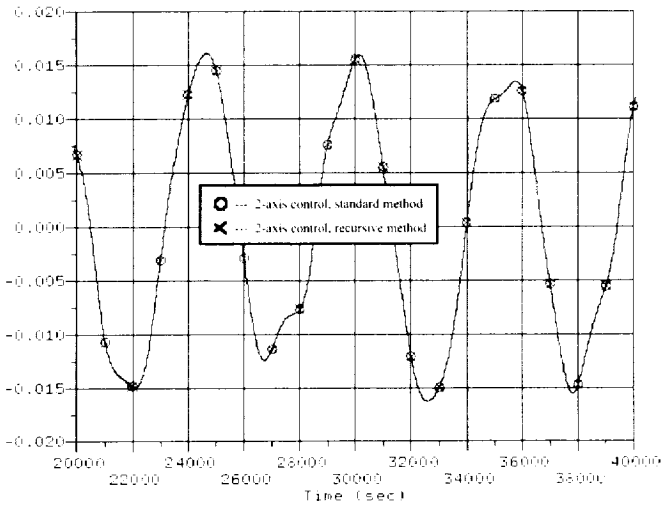

Case 2, Euler Angle, x-Axis (degrees)
0.0020
0.0015
0.0010
0.0005
0.0000
-0.0005
-0.0010
-0.0015
40000
42000
44000
46000
48000
50000
52000
54000
56000
58000
60000
Time (sec)
--- 2-axis control, standard method
--- 2-axis control, recursive method


Case 2, Euler Angle, x-Axis (degrees)
0.0008
0.0006
0.0004
0.0002
0.0000
-0.0002
-0.0004
-0.0006
0
50
100
150
200
250
300
350
400
450
500
Time (sec)
--- 2-axis control, standard method
--- 2-axis control, recursive method

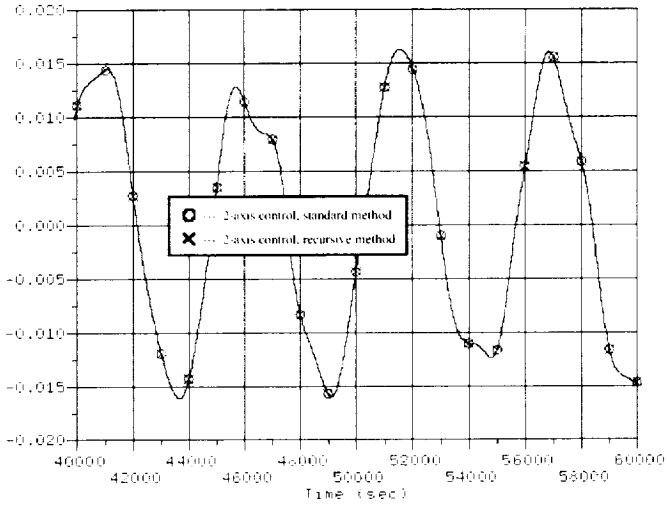

Case 2, Euler Angle, x-Axis (degrees)
x10^-2
0.130
0.129
0.128
0.127
0.126
0.125
0.124
0.123
0.122
0.121
59500
59550
59600
59650
59700
59750
59800
59850
59900
59950
60000
Time (sec)
--- 2-axis control, standard method
--- 2-axis control, recursive method


Case 2, Euler Angle, y-Axis (degrees)
0.03
0.02
0.01
0.00
-0.01
-0.02
-0.03
-0.04
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Time (sec)
--- 2-axis control, standard method
--- 2-axis control, recursive method

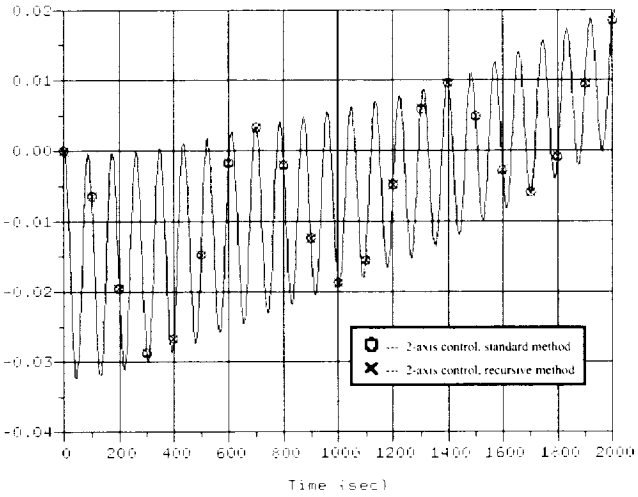
Case 2, Euler Angle, y-Axis (degrees)



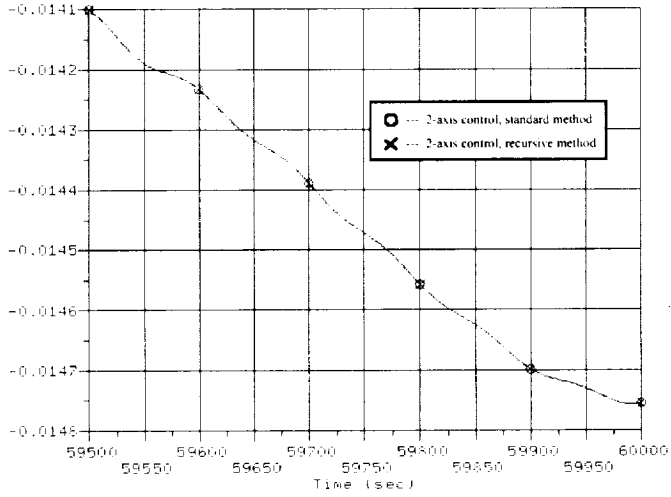
Case 2, Euler Angle, y-Axis (degrees)



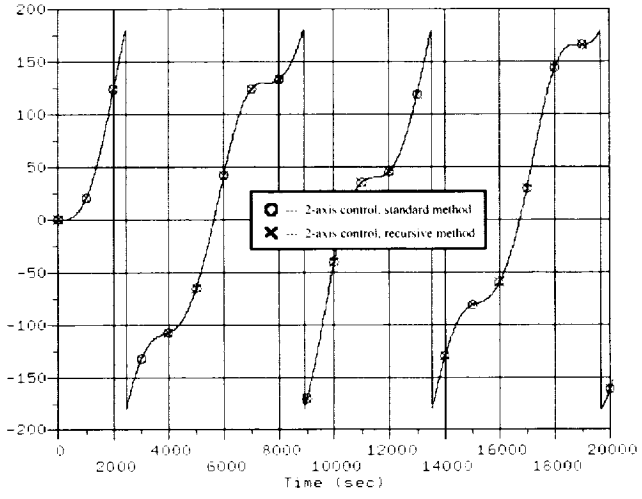
Case 2, Euler Angle, y-Axis (degrees)



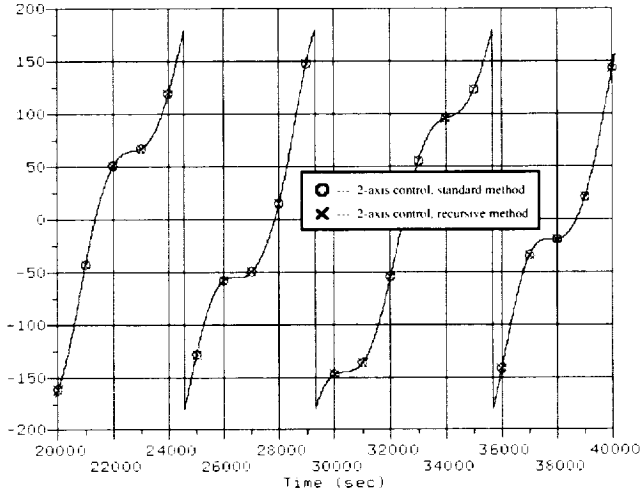
Case 2, Euler Angle, y-Axis (degrees)

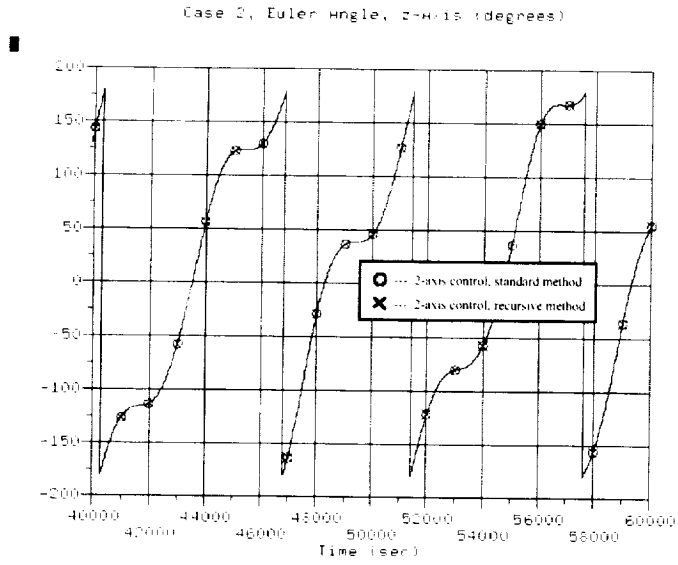


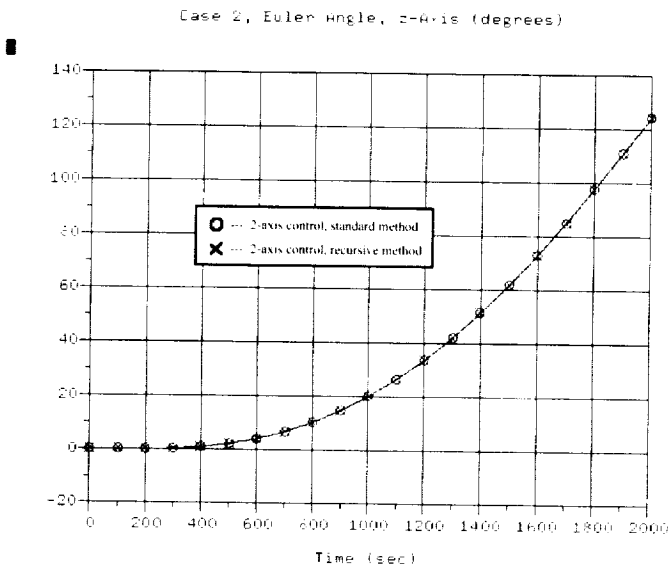
Case 2, Euler Angle, z-Axis (degrees)

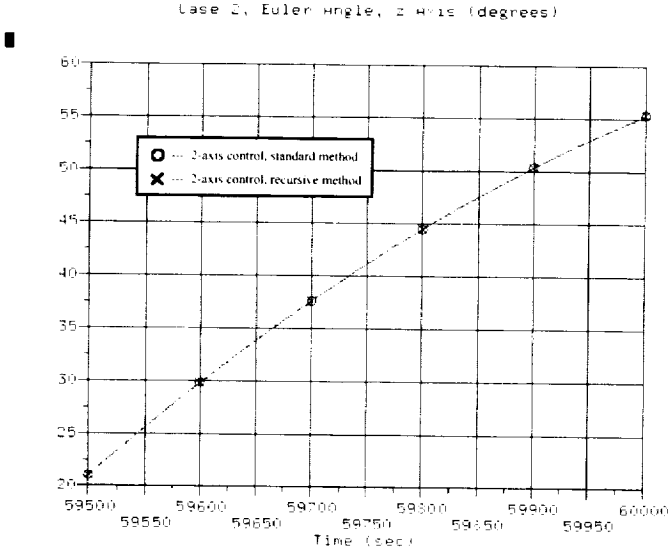


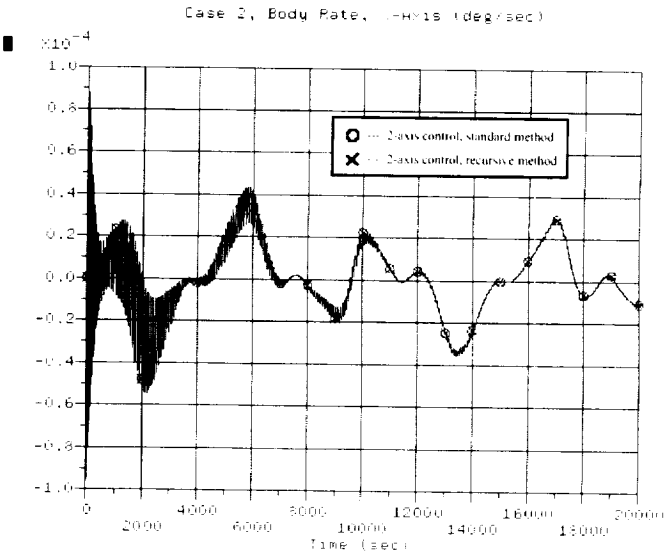
Case 2, Euler Angle, z-Axis (degrees)

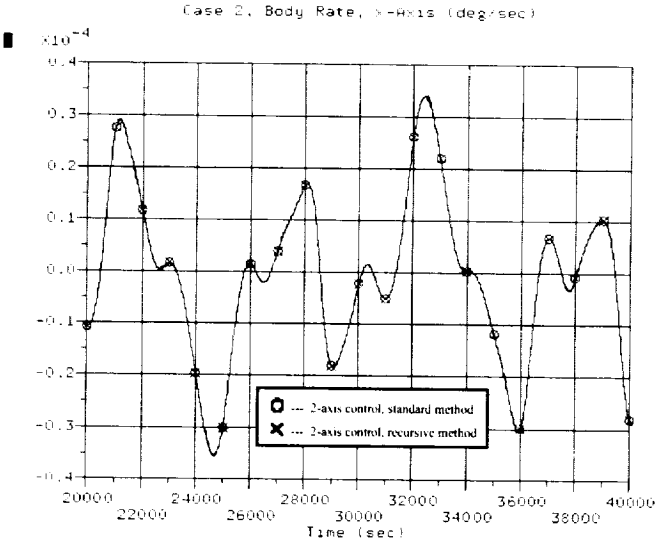


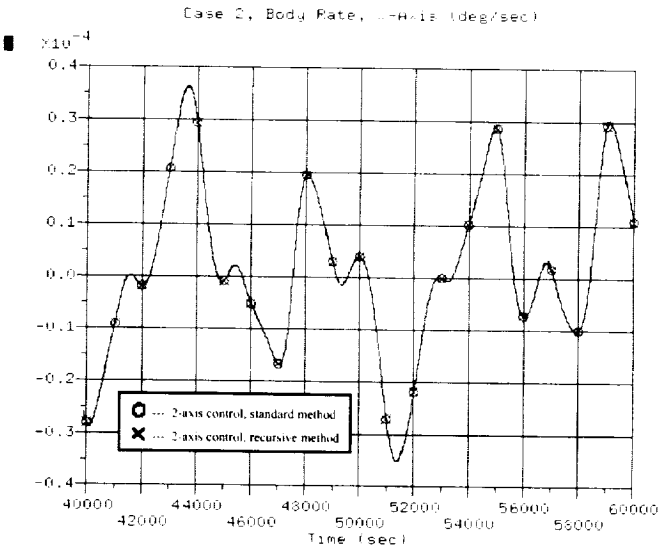

Case 2, Euler Angle, z-Axis (degrees)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method
Time (sec)

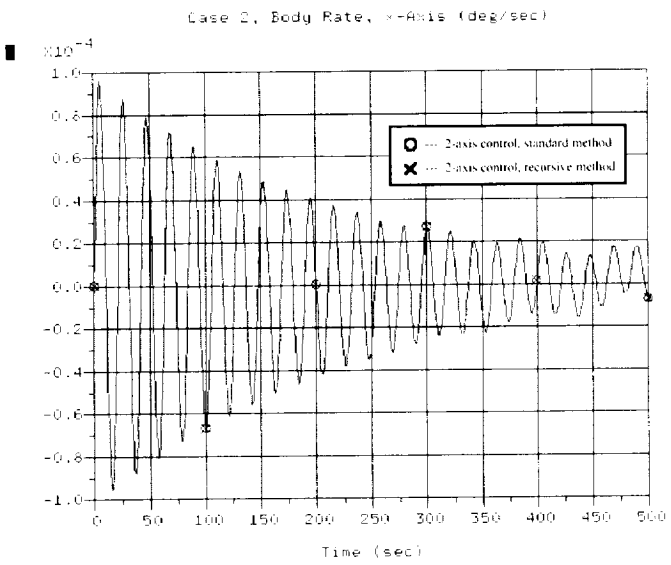

Case 2, Euler Angle, z-Axis (degrees)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method
Time (sec)

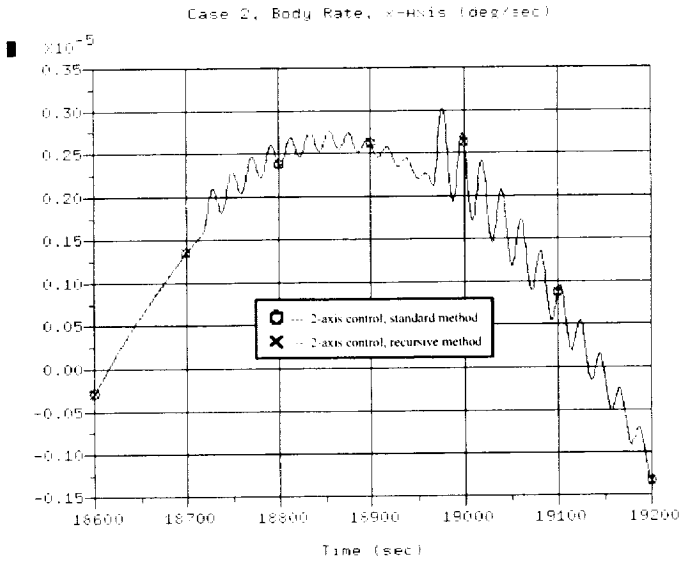

Case 2, Euler Angle, z-Axis (degrees)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method
Time (sec)

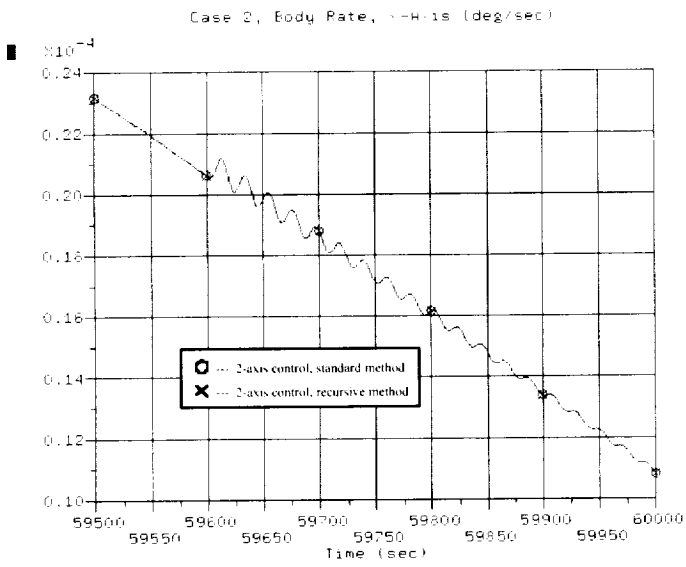

Case 2, Body Rate, x-Axis (deg/sec)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method
Time (sec)

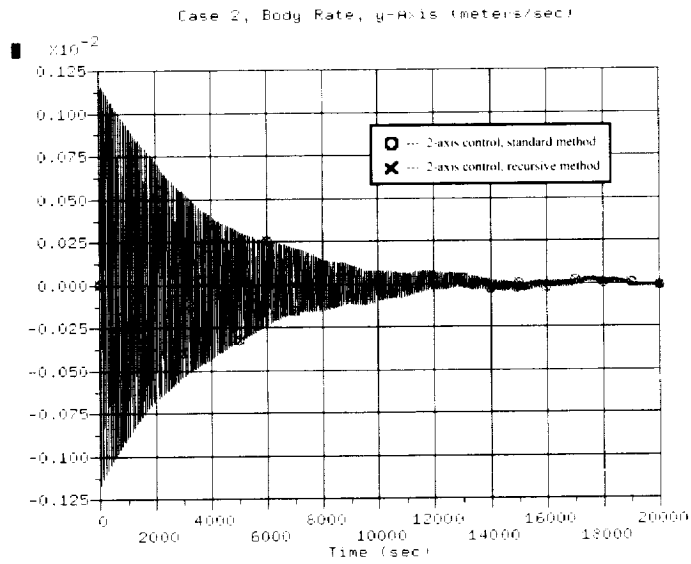

Case 2, Body Rate, x-Axis (deg/sec)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method
Time (sec)

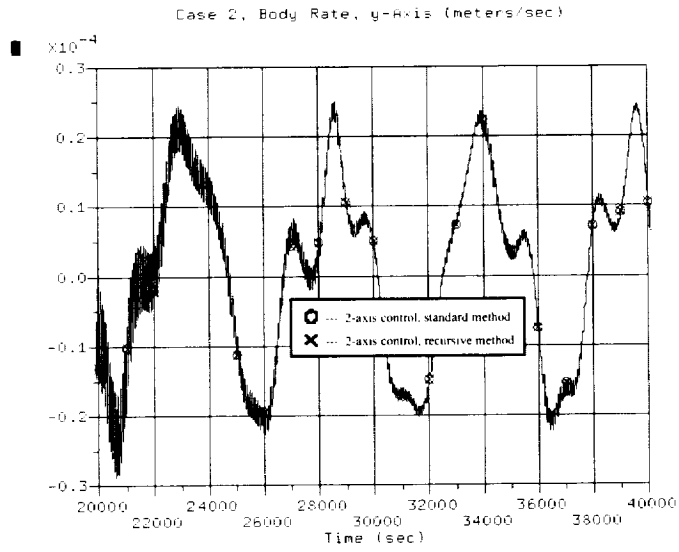

Case 2, Body Rate, x-Axis (deg/sec)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method
Time (sec)

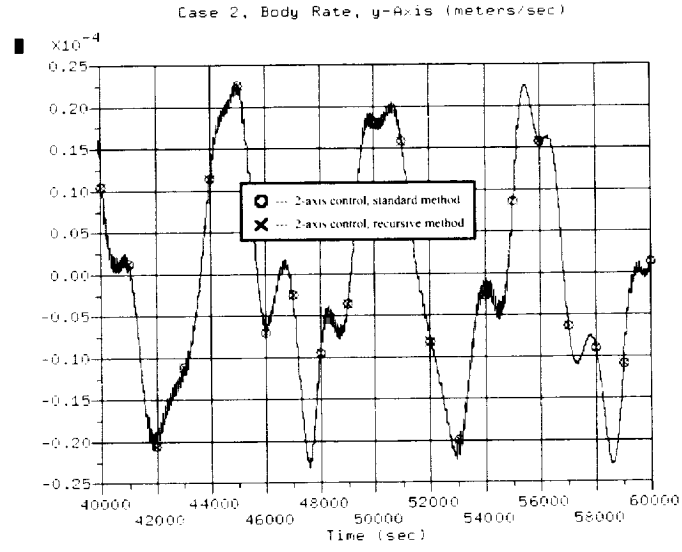

Case 2, Body Rate, x-Axis (deg/sec)
X10^-4
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
-0.4
-0.6
-0.8
-1.0
0
50
100
150
200
250
300
350
400
450
500
Time (sec)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method

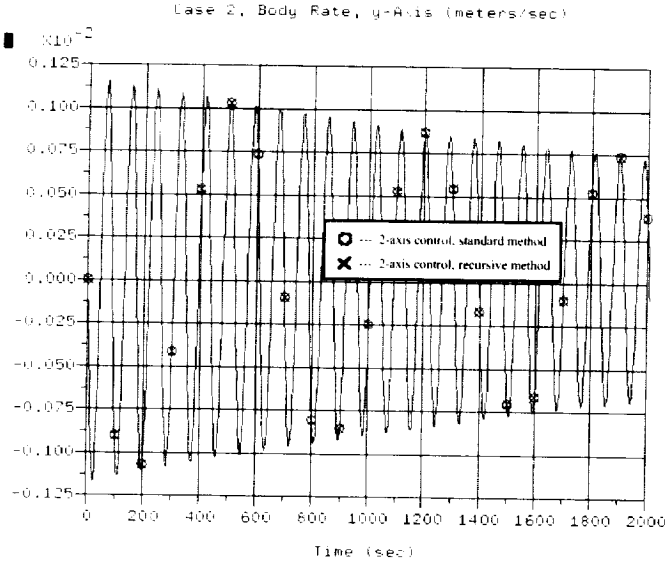

Case 2, Body Rate, x-Axis (deg/sec)
X10^-5
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
-0.05
-0.10
-0.15
18600
18700
18800
18900
19000
19100
19200
Time (sec)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method

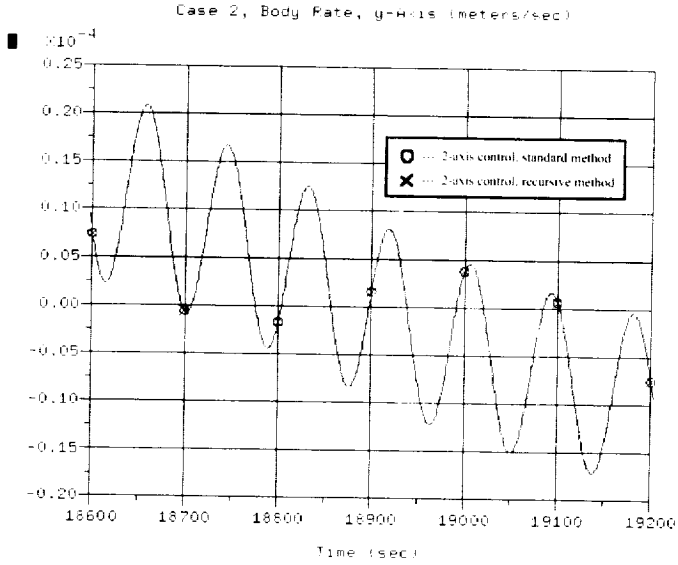

Case 2, Body Rate, x-Axis (deg/sec)
X10^-4
0.24
0.22
0.20
0.18
0.16
0.14
0.12
0.10
59500
59550
59600
59650
59700
59750
59800
59850
59900
59950
60000
Time (sec)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method

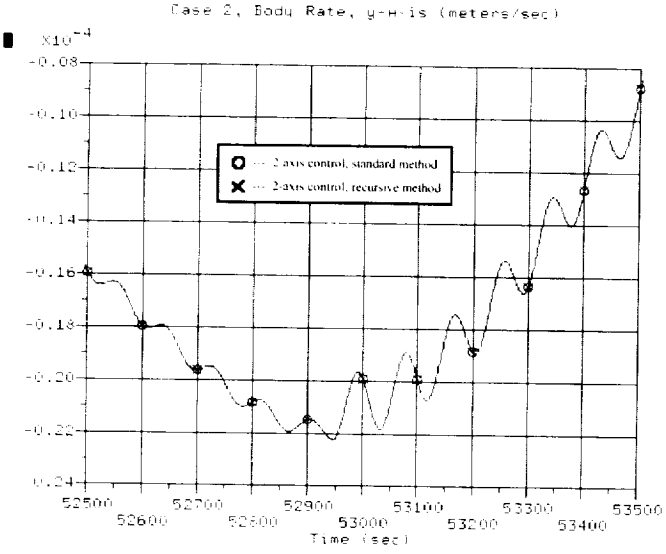

Case 2, Body Rate, y-Axis (meters/sec)
X10^-2
0.125
0.100
0.075
0.050
0.025
0.000
-0.025
-0.050
-0.075
-0.100
-0.125
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Time (sec)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method

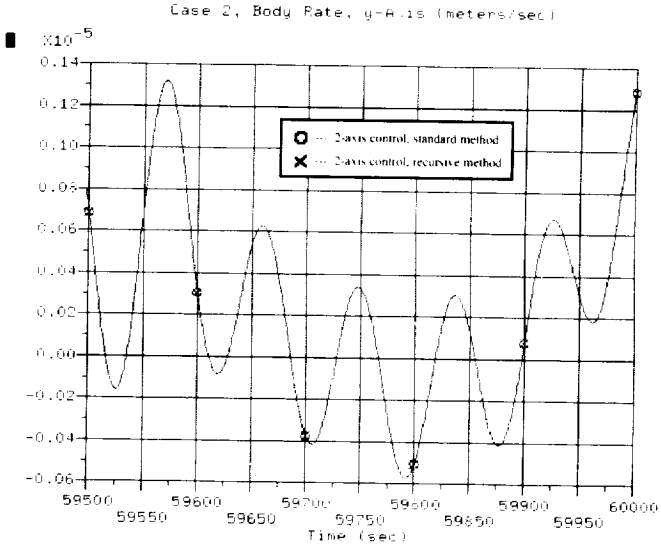

Case 2, Body Rate, y-Axis (meters/sec)
X10^-4
0.3
0.2
0.1
0.0
-0.1
-0.2
-0.3
20000
22000
24000
26000
28000
30000
32000
34000
36000
38000
40000
Time (sec)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method

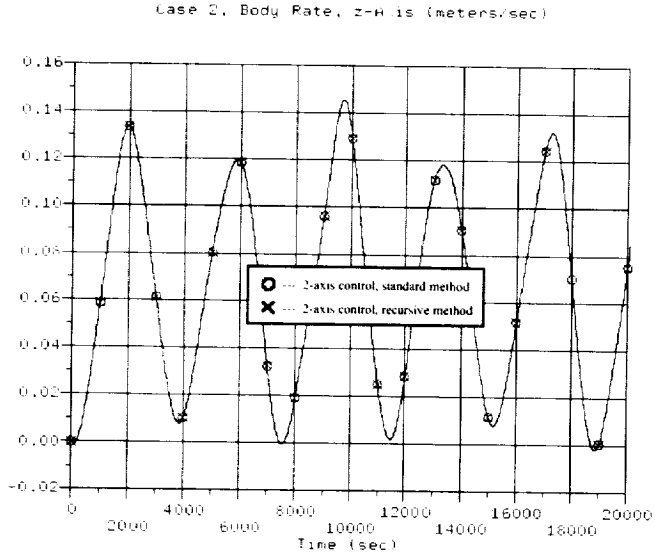

Case 2, Body Rate, y-Axis (meters/sec)
X10^-4
0.25
0.20
0.15
0.10
0.05
0.00
-0.05
-0.10
-0.15
-0.20
-0.25
40000
42000
44000
46000
48000
50000
52000
54000
56000
58000
60000
Time (sec)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method

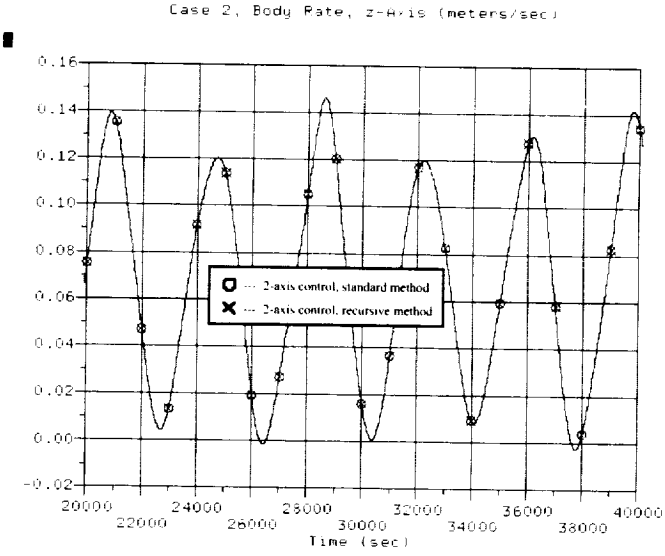

Case 2, Body Rate, y-Axis (meters/sec)
X10^-2
0.125
0.100
0.075
0.050
0.025
0.000
-0.025
-0.050
-0.075
-0.100
-0.125
0
200
400
600
800
1000
1200
1400
1600
1800
2000
Time (sec)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method

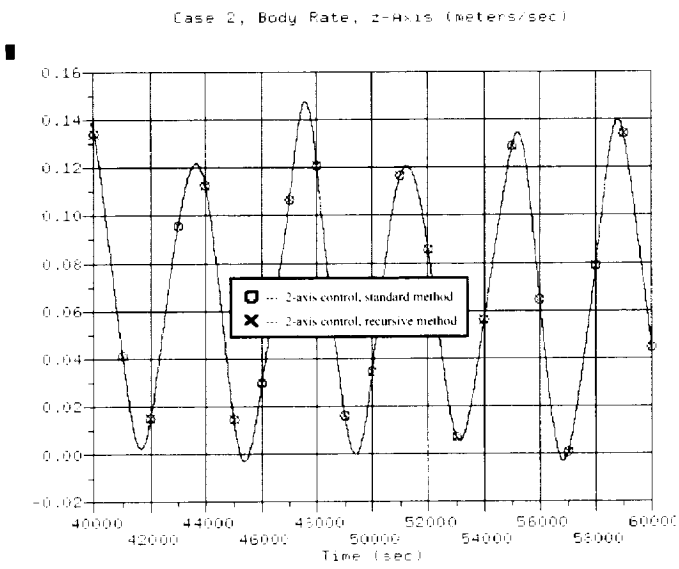

Case 2, Body Rate, y-Axis (meters/sec)
X10^-4
0.25
0.20
0.15
0.10
0.05
0.00
-0.05
-0.10
-0.15
-0.20
18600
18700
18800
18900
19000
19100
19200
Time (sec)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method

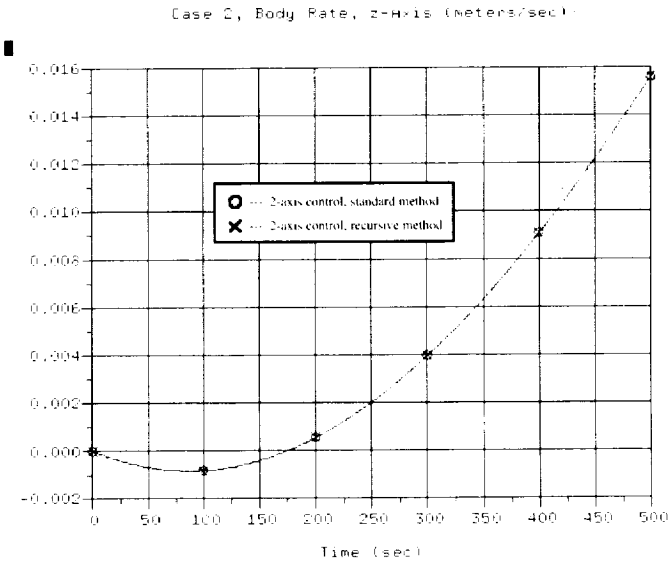

Case 2, Body Rate, y-Axis (meters/sec)
X10^-4
-0.08
-0.10
-0.12
-0.14
-0.16
-0.18
-0.20
-0.22
-0.24
52500
52600
52700
52800
52900
53000
53100
53200
53300
53400
53500
Time (sec)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method

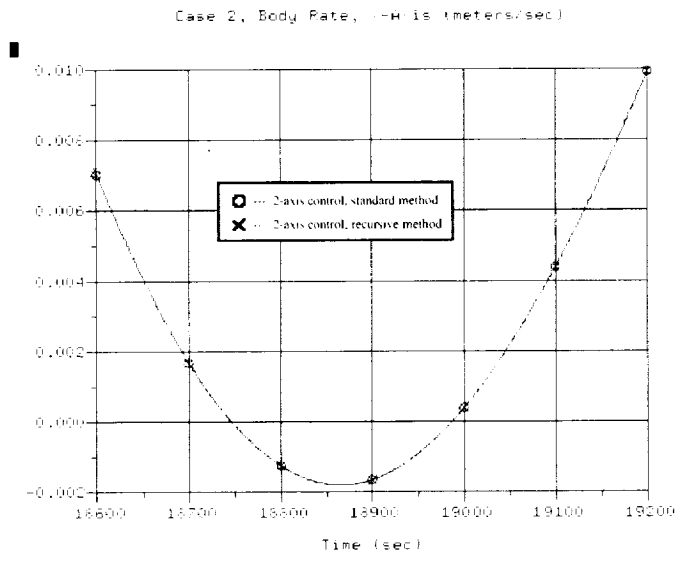

Case 2, Body Rate, y-Axis (meters/sec)
X10^-5
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
-0.02
-0.04
-0.06
59500
59550
59600
59650
59700
59750
59800
59850
59900
59950
60000
Time (sec)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method

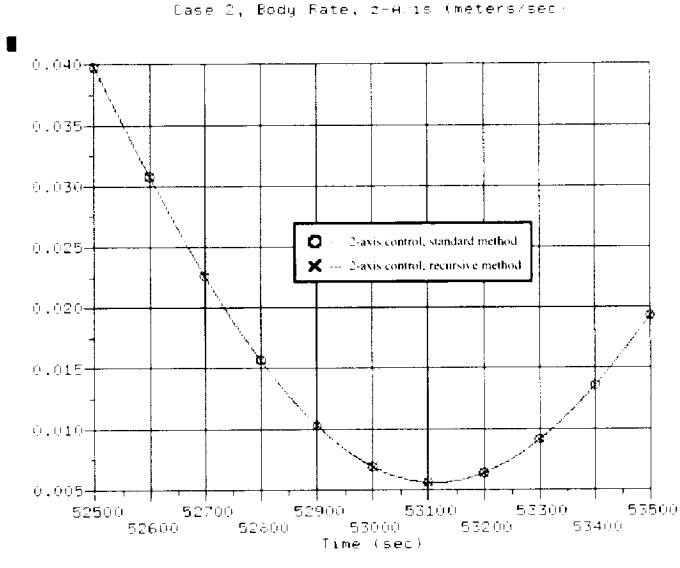

Case 2, Body Rate, z-Axis (meters/sec)
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
-0.02
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Time (sec)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method

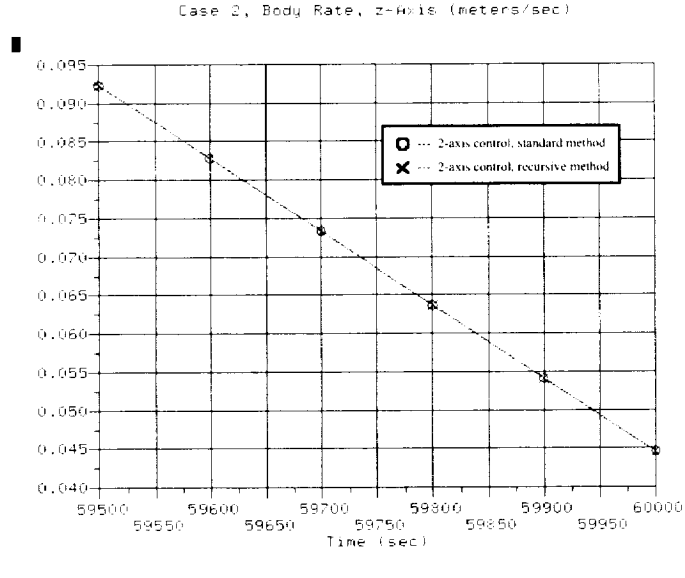

Case 2, Body Rate, z-Axis (meters/sec)
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
-0.02
20000
22000
24000
26000
28000
30000
32000
34000
36000
38000
40000
Time (sec)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method

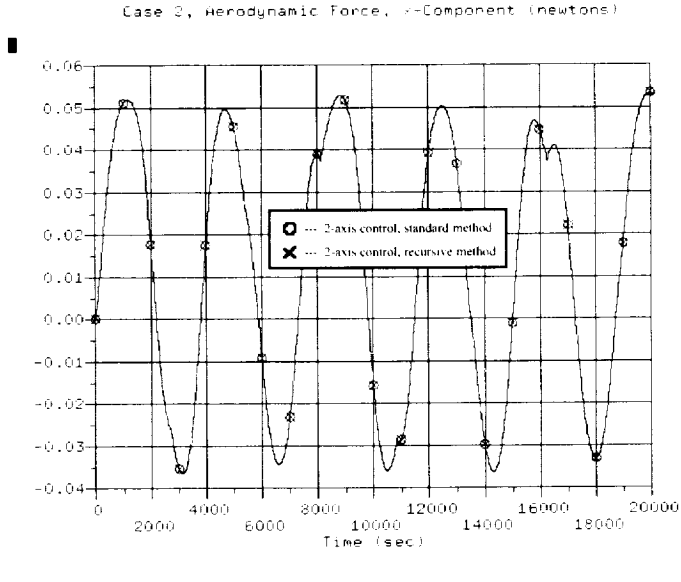

Case 2, Body Rate, z-Axis (meters/sec)
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
-0.02
40000
42000
44000
46000
48000
50000
52000
54000
56000
58000
60000
Time (sec)
O ··· 2-axis control, standard method
X ··· 2-axis control, recursive method

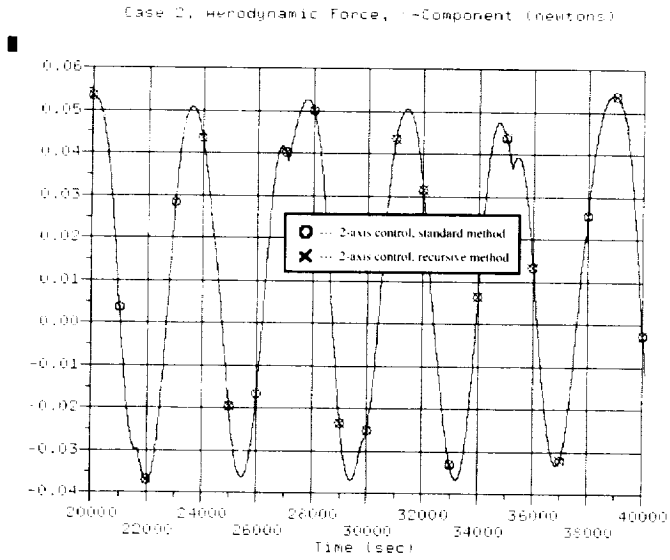

Case 2, Body Rate, z-Axis (meters/sec)
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
-0.002
0
50
100
150
200
250
300
350
400
450
500
Time (sec)
O ··· 2-axis control, standard method
X ··· 2-axis control, recursive method

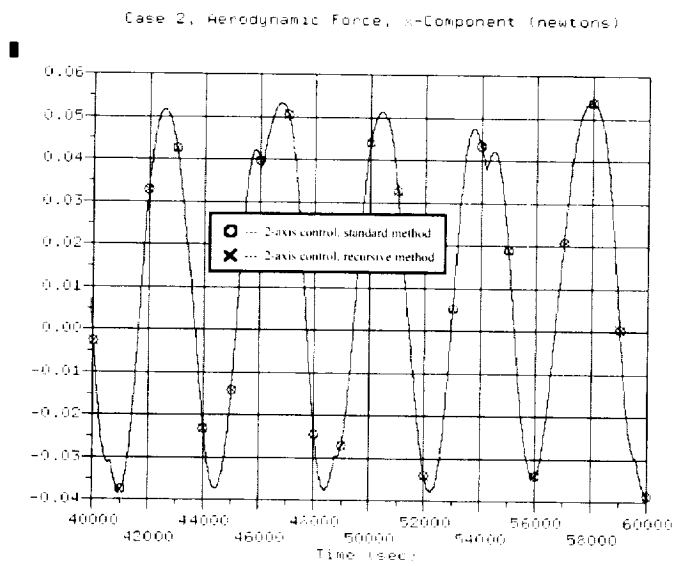

Case 2, Body Rate, z-Axis (meters/sec)
0.010
0.008
0.006
0.004
0.002
0.000
-0.002
18600
18700
18800
18900
19000
19100
19200
Time (sec)
O ··· 2-axis control, standard method
X ··· 2-axis control, recursive method

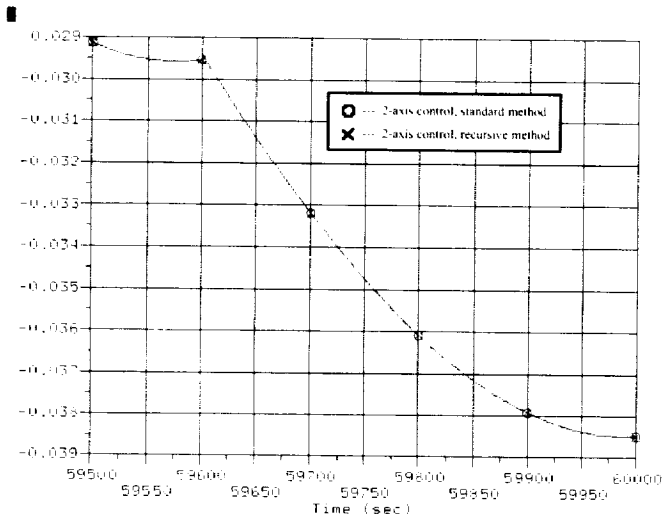

Case 2, Body Rate, z-Axis (meters/sec)
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
52500
52600
52700
52800
52900
53000
53100
53200
53300
53400
53500
Time (sec)
O ··· 2-axis control, standard method
X ··· 2-axis control, recursive method

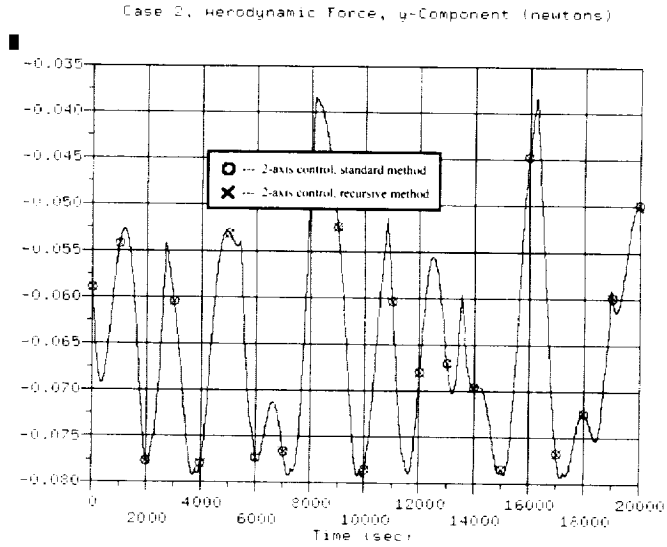

Case 2, Body Rate, z-Axis (meters/sec)
0.095
0.090
0.085
0.080
0.075
0.070
0.065
0.060
0.055
0.050
0.045
0.040
59500
59550
59600
59650
59700
59750
59800
59850
59900
59950
60000
Time (sec)
O ··· 2-axis control, standard method
X ··· 2-axis control, recursive method

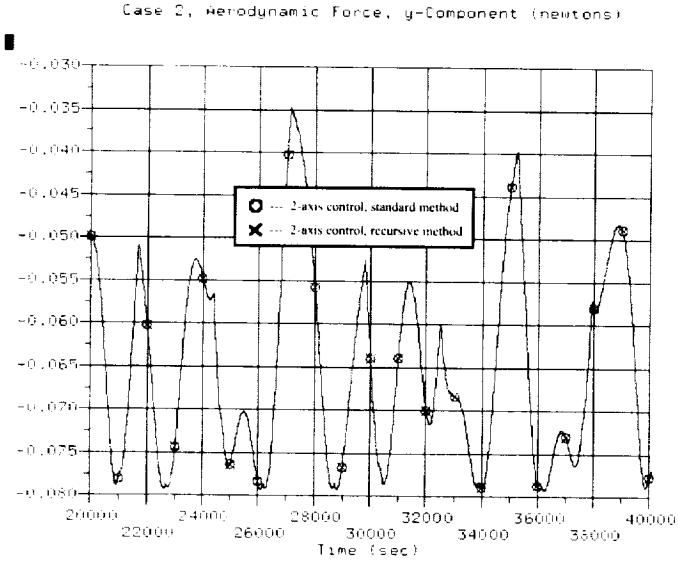

Case 2, Aerodynamic Force, x-Component (newtons)
0.06
0.05
0.04
0.03
0.02
0.01
0.00
-0.01
-0.02
-0.03
-0.04
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Time (sec)
O ··· 2-axis control, standard method
X ··· 2-axis control, recursive method

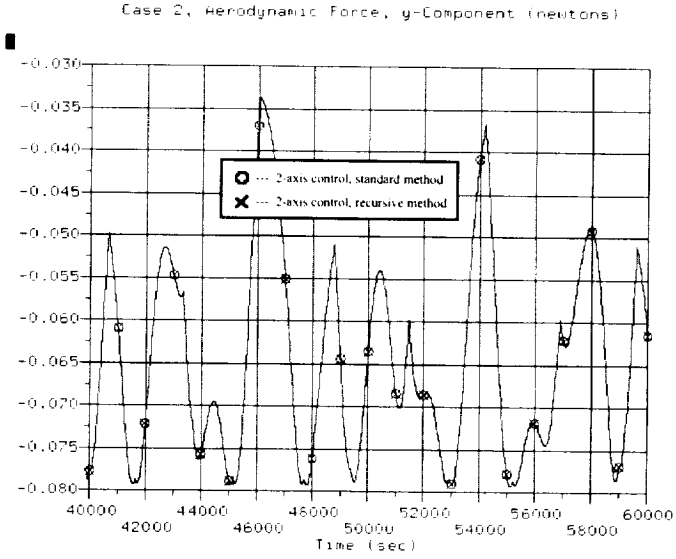

Case 2, Aerodynamic Force, x-Component (newtons)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method
Time (sec)


Case 2, Aerodynamic Force, x-Component (newtons)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method
Time (sec)

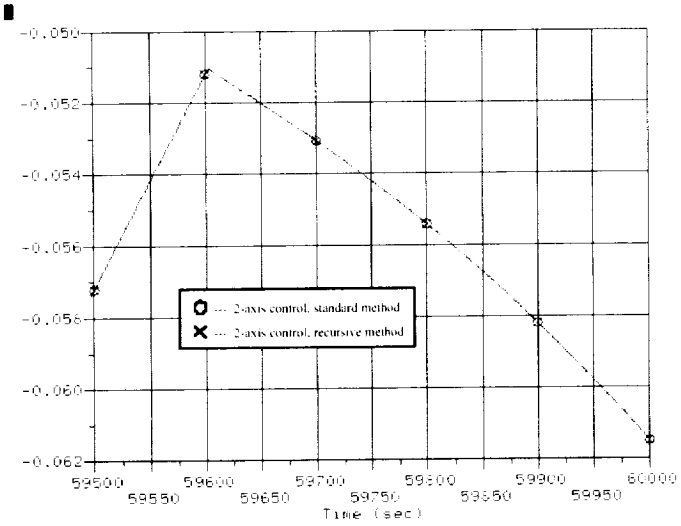

Case 2, Aerodynamic Force, z-Component (newtons)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method
Time (sec)


Case 2, Aerodynamic Force, y-Component (newtons)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method
Time (sec)

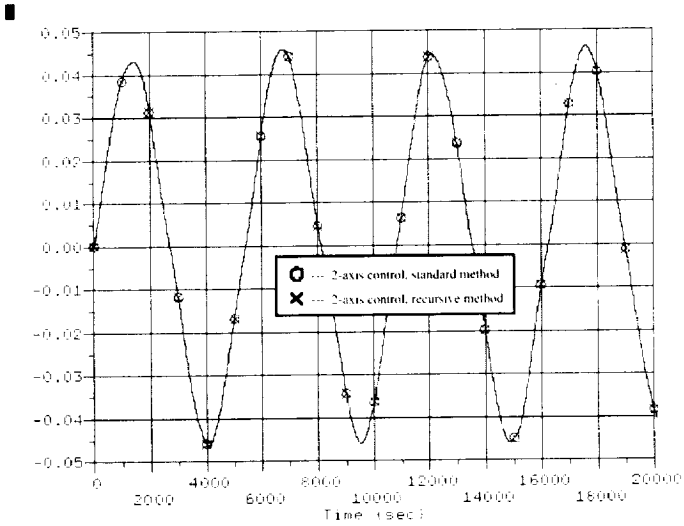

Case 2, Aerodynamic Force, y-Component (newtons)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method
Time (sec)


Case 2, Aerodynamic Force, y-Component (newtons)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method
Time (sec)

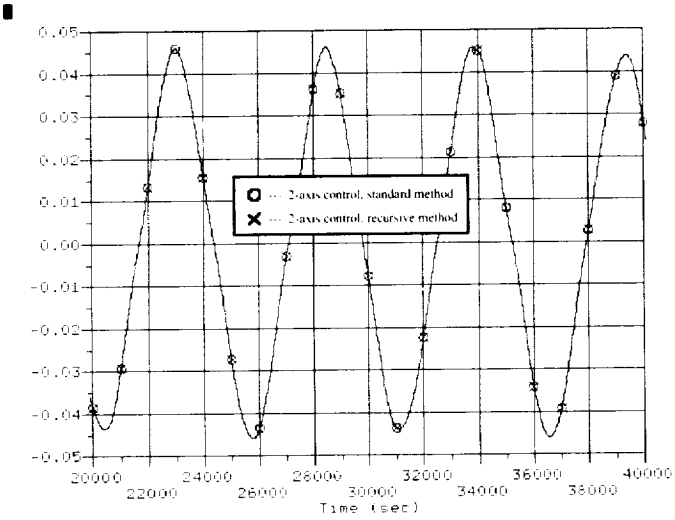
Case 2, Aerodynamic Force, y-Component (newtons)



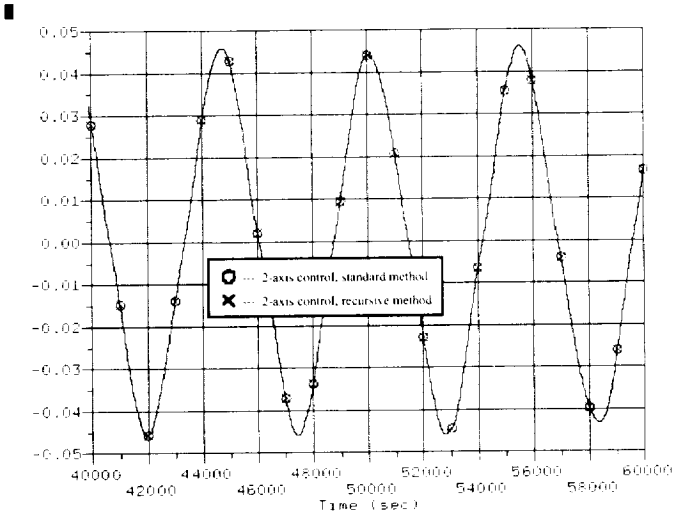
Case 2, Aerodynamic Force, z-Component (newtons)



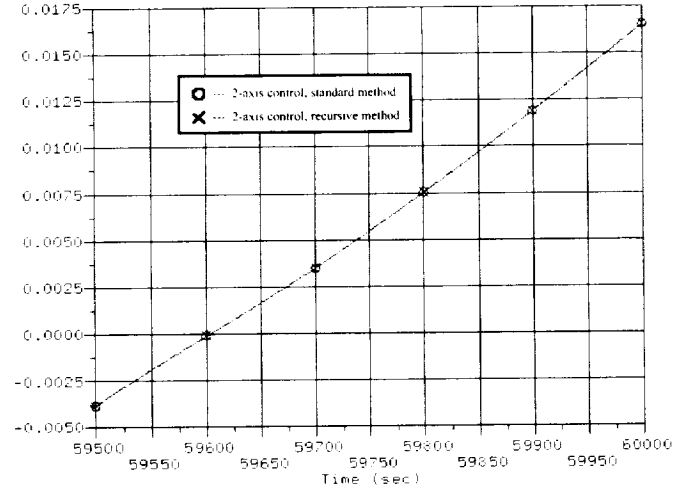
Case 2, Aerodynamic Force, z-Component (newtons)



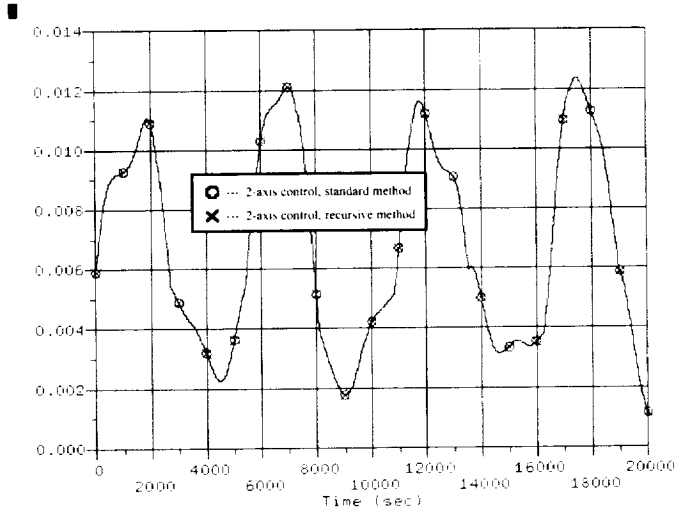
Case 2, Aerodynamic Force, z-Component (newtons)

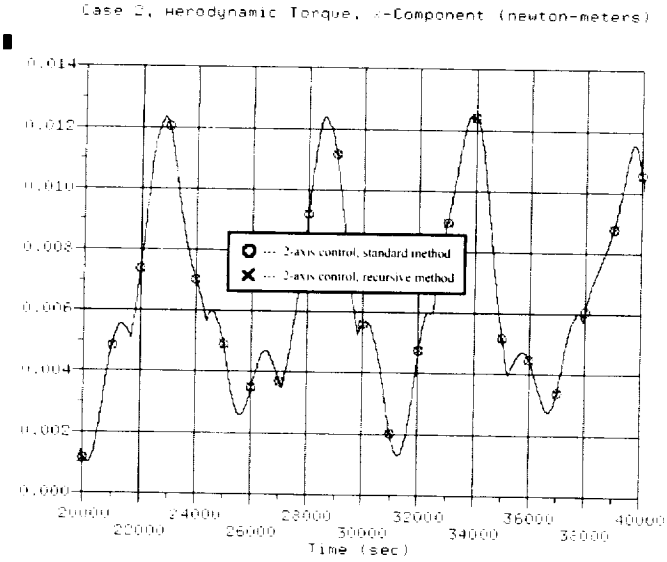


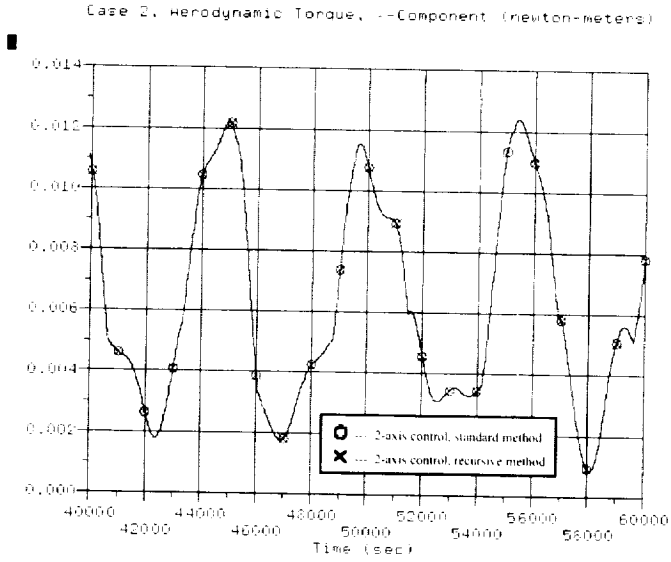
Case 2, Aerodynamic Force, z-Component (newtons)

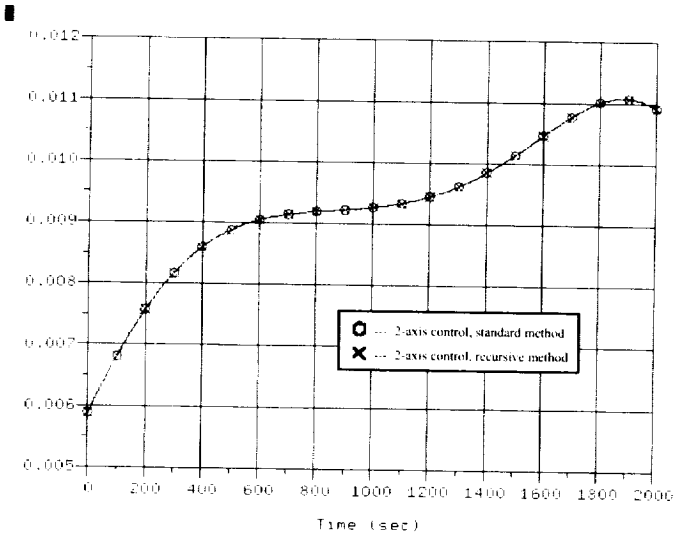


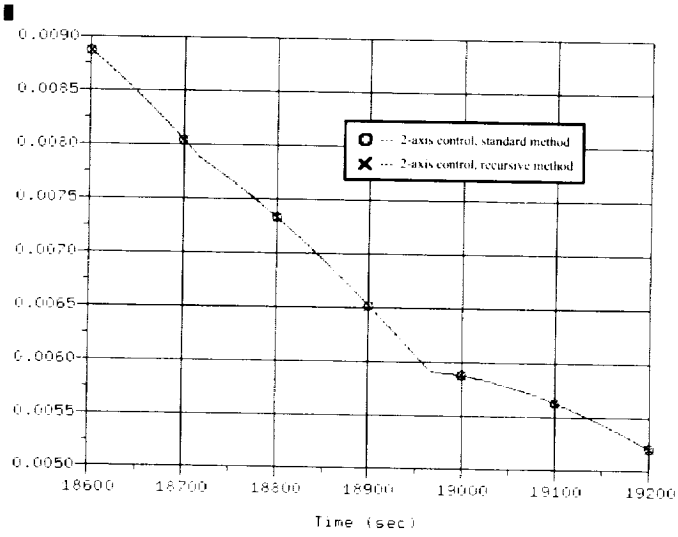
Case 2, Aerodynamic Torque, z-Component (newton-meters)

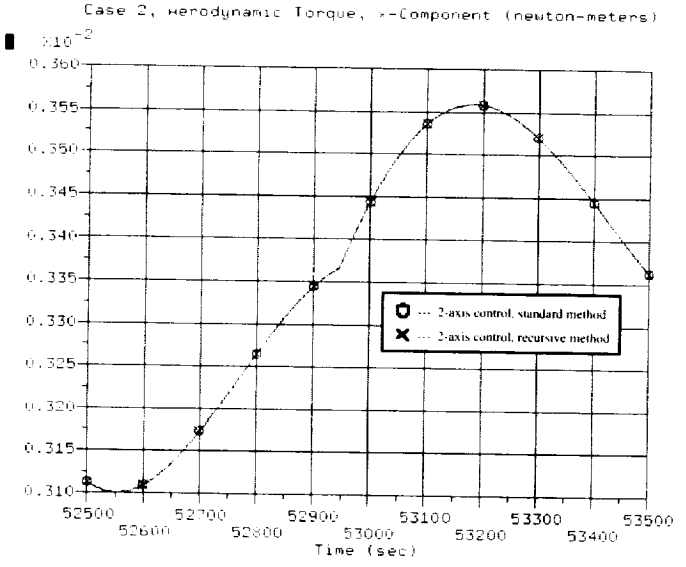


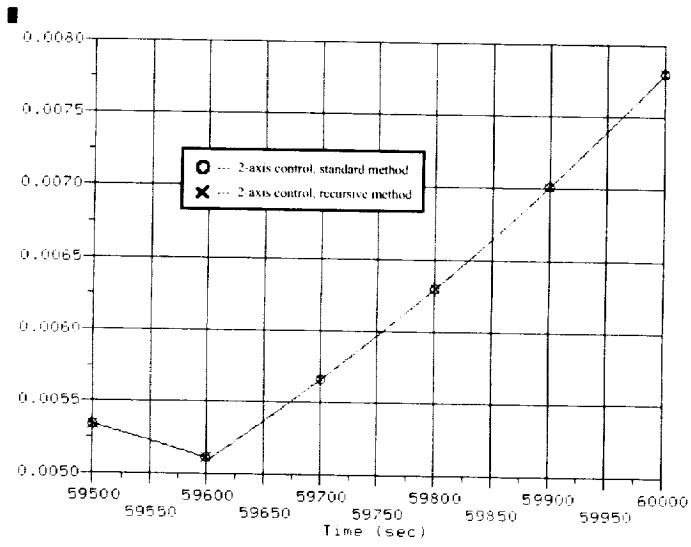

Case 2, Aerodynamic Torque, x-Component (newton-meters)
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
20000
22000
24000
26000
28000
30000
32000
34000
36000
38000
40000
Time (sec)
O ... 2-axis control, standard method
X ... 2-axis control, recursive method

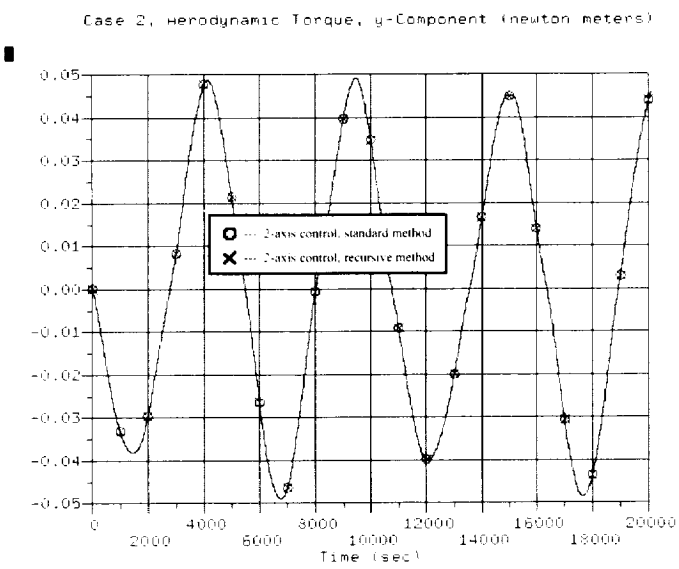

Case 2, Aerodynamic Torque, x-Component (newton-meters)
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
40000
42000
44000
46000
48000
50000
52000
54000
56000
58000
60000
Time (sec)
O ... 2-axis control, standard method
X ... 2-axis control, recursive method

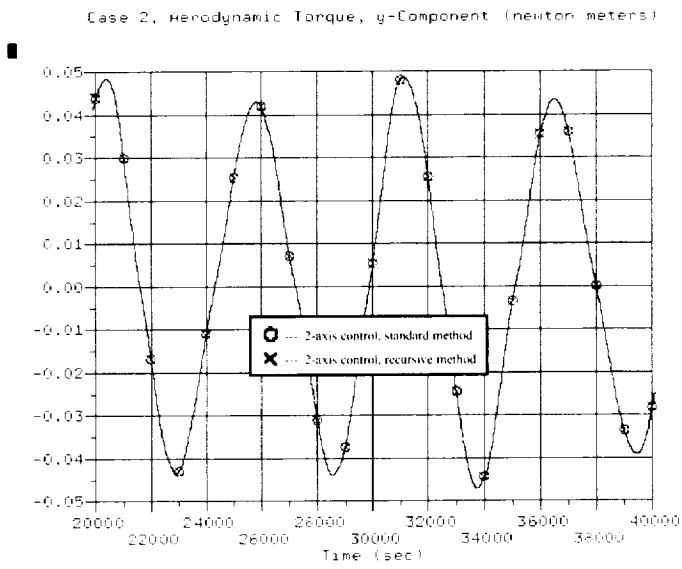

Case 2, Aerodynamic Torque, x-Component (newton-meters)
0.012
0.011
0.010
0.009
0.008
0.007
0.006
0.005
0
200
400
600
800
1000
1200
1400
1600
1800
2000
Time (sec)
O ... 2-axis control, standard method
X ... 2-axis control, recursive method

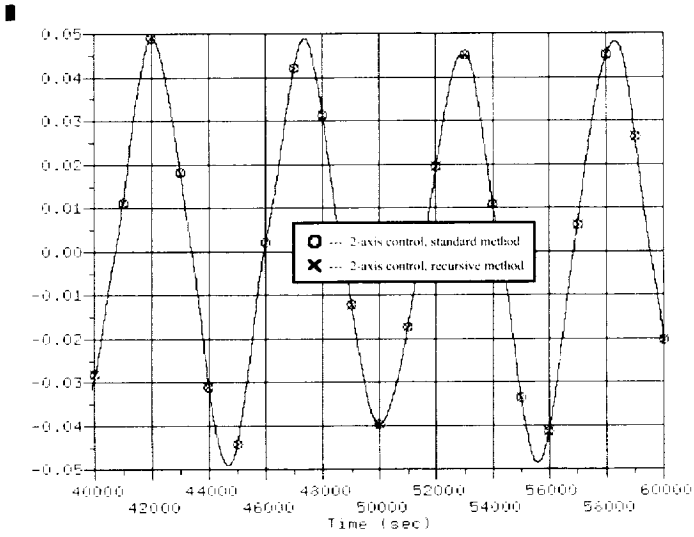

Case 2, Aerodynamic Torque, x-Component (newton-meters)
0.0090
0.0085
0.0080
0.0075
0.0070
0.0065
0.0060
0.0055
0.0050
18600
18700
18800
18900
19000
19100
19200
Time (sec)
O ... 2-axis control, standard method
X ... 2-axis control, recursive method

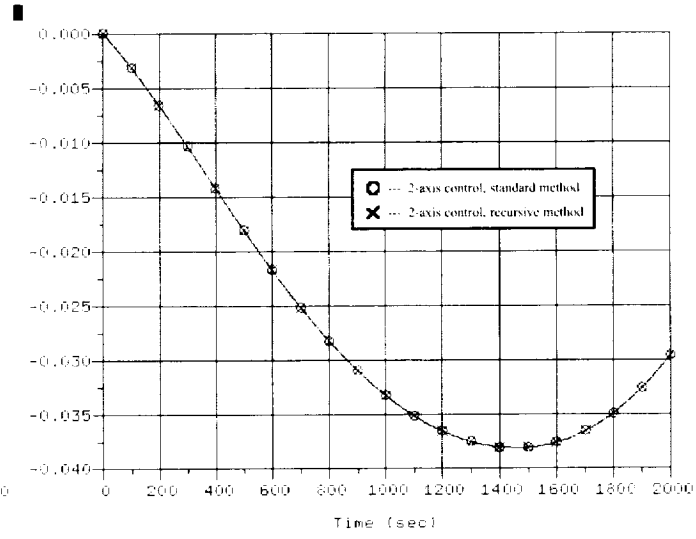

Case 2, Aerodynamic Torque, x-Component (newton-meters)
x10^-2
0.360
0.355
0.350
0.345
0.340
0.335
0.330
0.325
0.320
0.315
0.310
52500
52600
52700
52800
52900
53000
53100
53200
53300
53400
53500
Time (sec)
O ... 2-axis control, standard method
X ... 2-axis control, recursive method

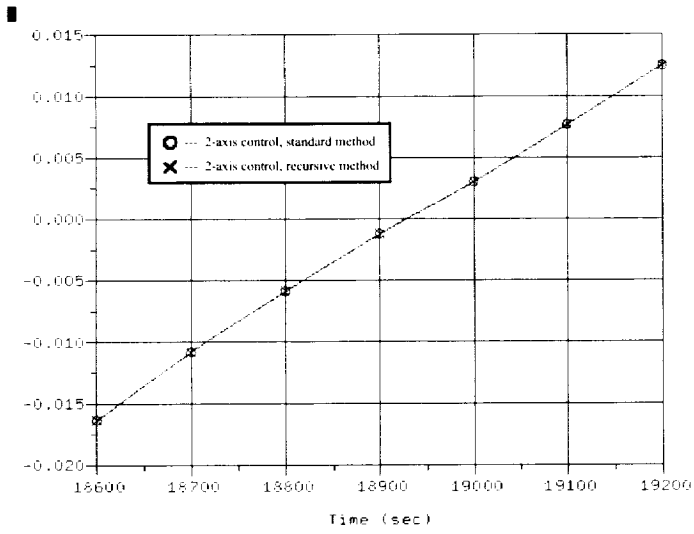

Case 2, Aerodynamic Torque, x-Component (newton-meters)
0.0080
0.0075
0.0070
0.0065
0.0060
0.0055
0.0050
59500
59550
59600
59650
59700
59750
59800
59850
59900
59950
60000
Time (sec)
O ... 2-axis control, standard method
X ... 2-axis control, recursive method

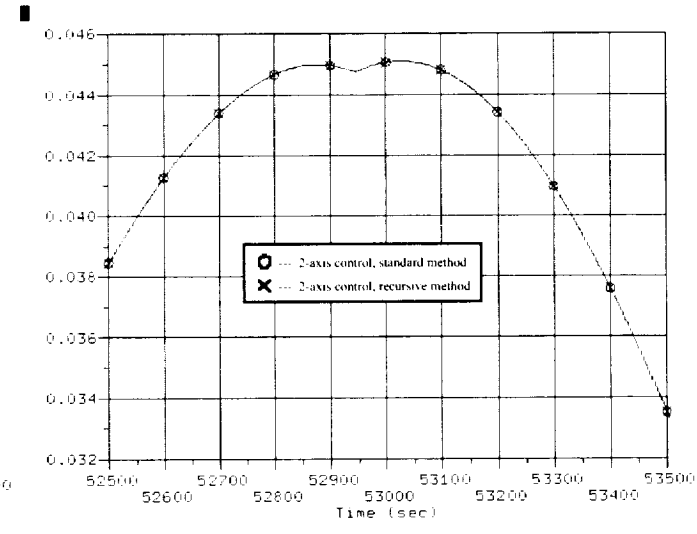

Case 2, Aerodynamic Torque, y-Component (newton meters)

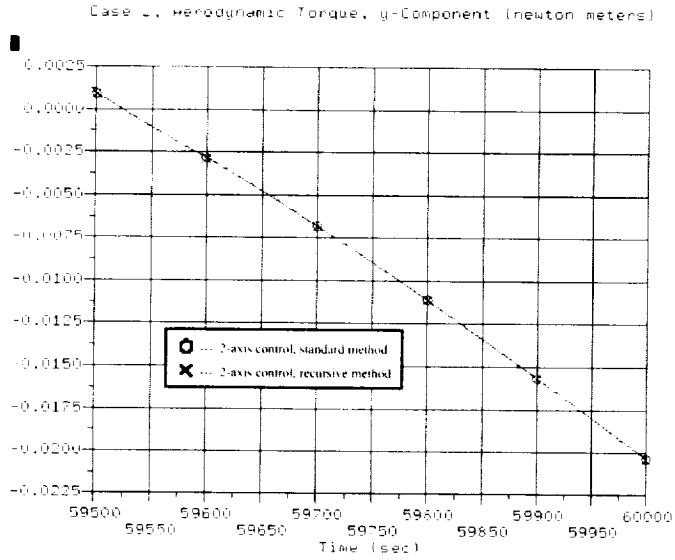

Case 2, Aerodynamic Torque, y-Component (newton meters)

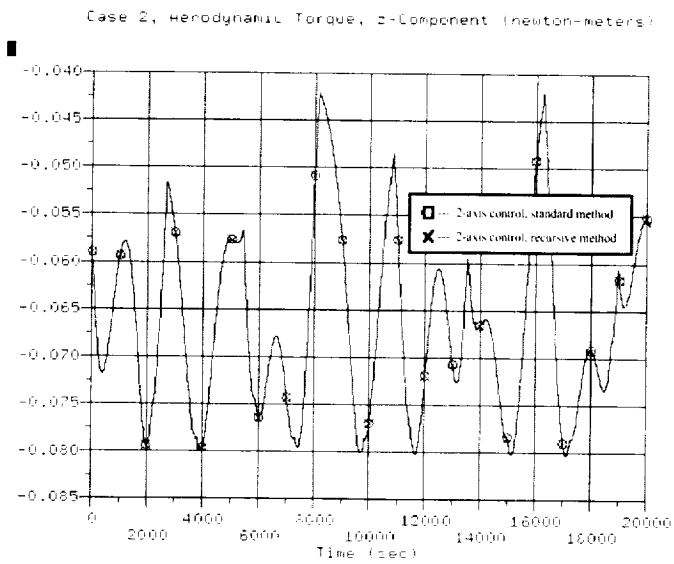

Case 2, Aerodynamic Torque, y-Component (newton meters)

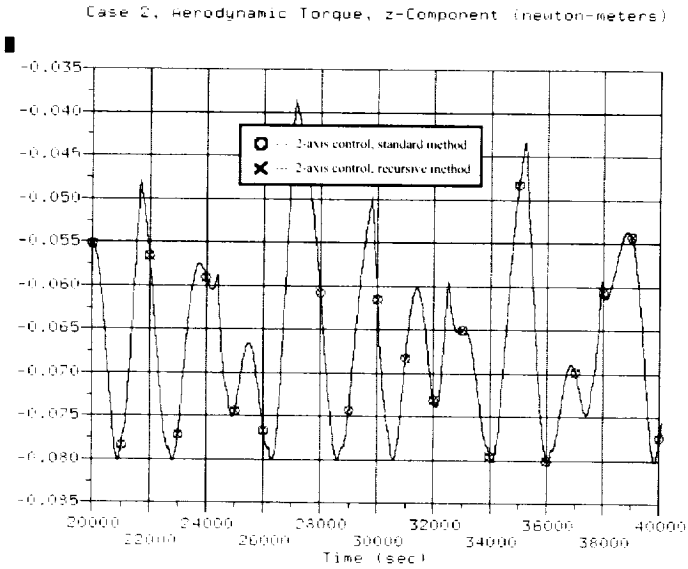

Case 2, Aerodynamic Torque, y-Component (newton meters)

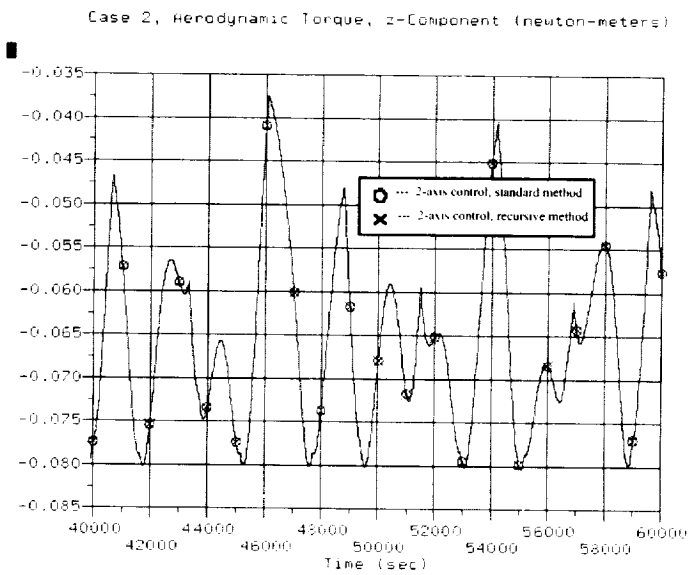

Case 2, Aerodynamic Torque, y-Component (newton meters)

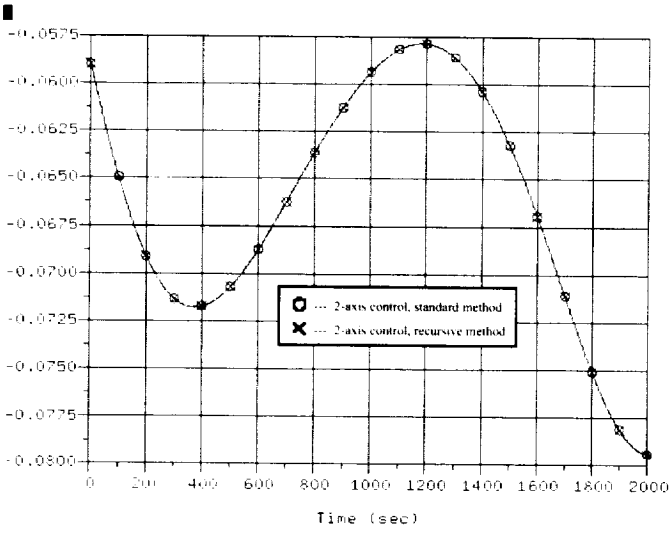

Case 2, Aerodynamic Torque, y-Component (newton meters)

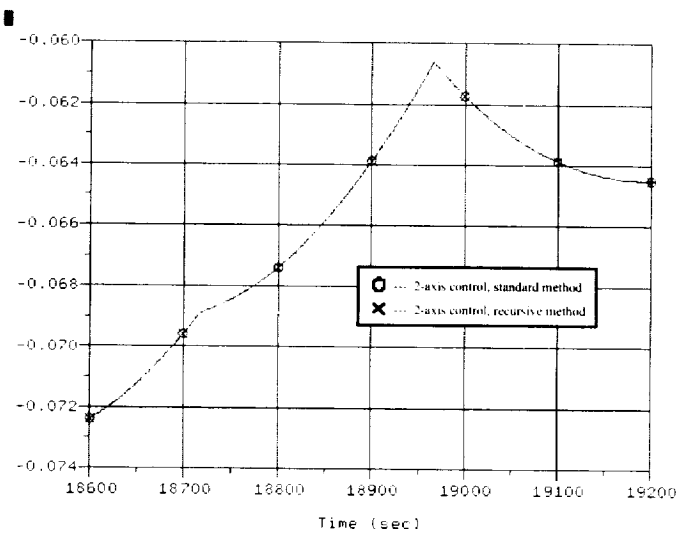

Case 2, Aerodynamic Torque, y-Component (newton meters)
-0.0025
0.0000
-0.0025
-0.0050
-0.0075
-0.0100
-0.0125
-0.0150
-0.0175
-0.0200
-0.0225
59500
59550
59600
59650
59700
59750
59800
59850
59900
59950
60000
Time (sec)
--- 2-axis control, standard method
--- 2-axis control, recursive method

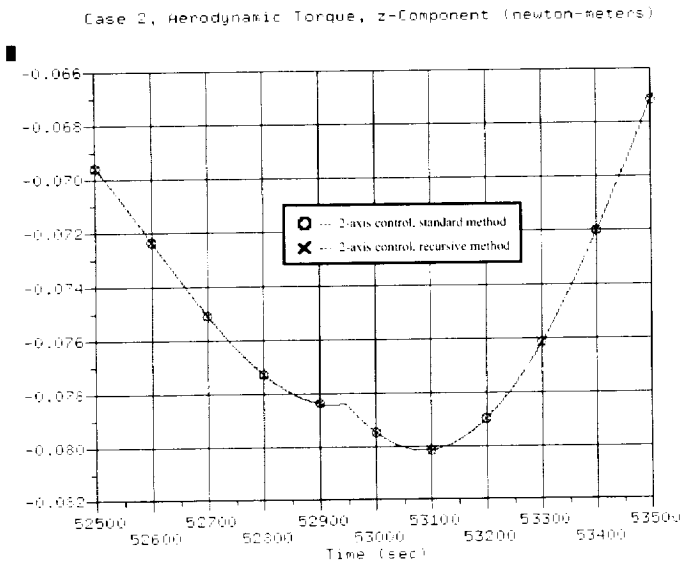

Case 2, Aerodynamic Torque, z-Component (newton-meters)
-0.040
-0.045
-0.050
-0.055
-0.060
-0.065
-0.070
-0.075
-0.080
-0.085
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Time (sec)
--- 2-axis control, standard method
--- 2-axis control, recursive method

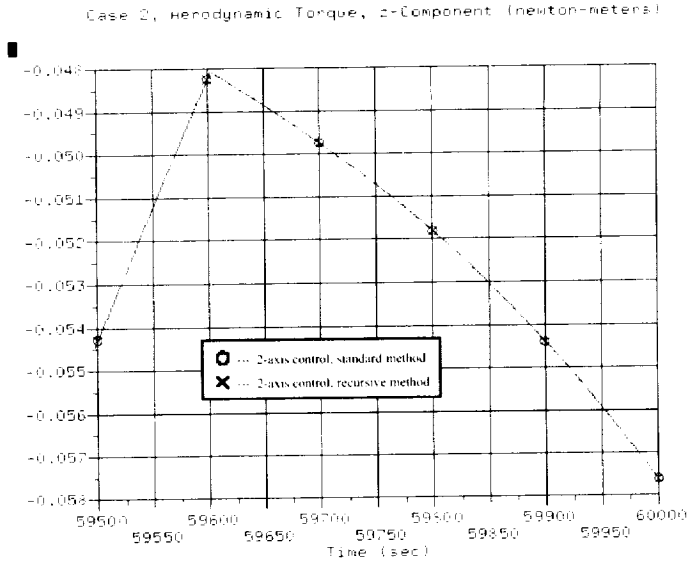

Case 2, Aerodynamic Torque, z-Component (newton-meters)
-0.035
-0.040
-0.045
-0.050
-0.055
-0.060
-0.065
-0.070
-0.075
-0.080
-0.085
20000
22000
24000
26000
28000
30000
32000
34000
36000
38000
40000
Time (sec)
--- 2-axis control, standard method
--- 2-axis control, recursive method


Case 2, Aerodynamic Torque, z-Component (newton-meters)
-0.035
-0.040
-0.045
-0.050
-0.055
-0.060
-0.065
-0.070
-0.075
-0.080
-0.085
40000
42000
44000
46000
48000
50000
52000
54000
56000
58000
60000
Time (sec)
--- 2-axis control, standard method
--- 2-axis control, recursive method

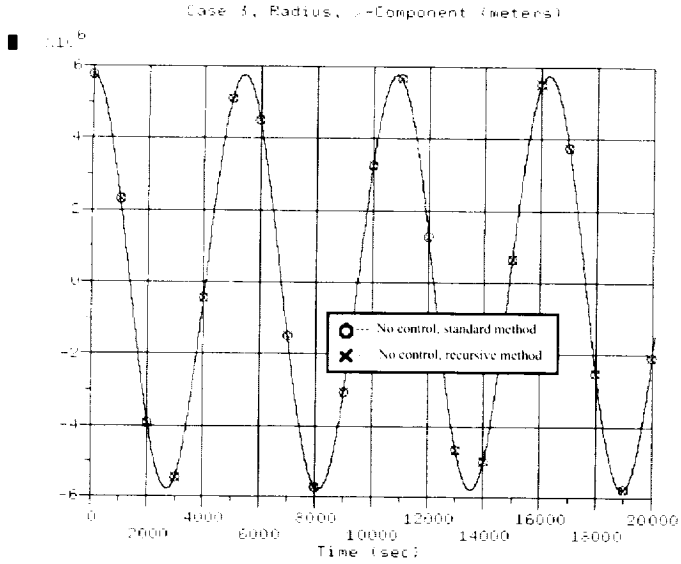

Case 2, Aerodynamic Torque, z-Component (newton-meters)
-0.0575
-0.0600
-0.0625
-0.0650
-0.0675
-0.0700
-0.0725
-0.0750
-0.0775
-0.0800
0
200
400
600
800
1000
1200
1400
1600
1800
2000
Time (sec)
--- 2-axis control, standard method
--- 2-axis control, recursive method

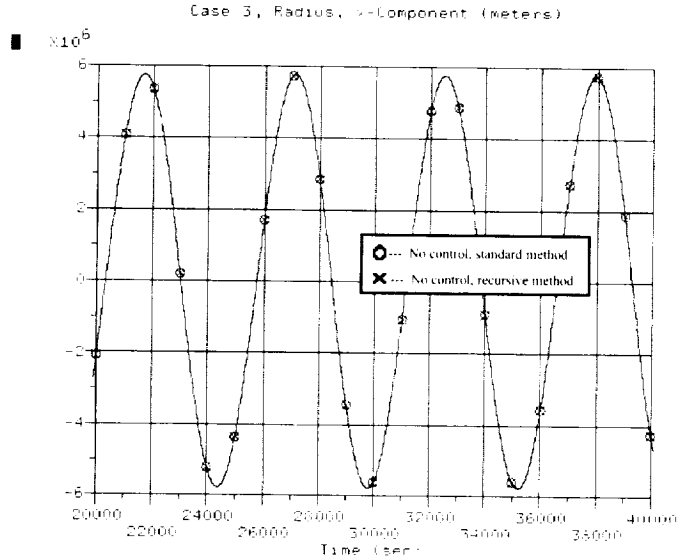

Case 2, Aerodynamic Torque, z-Component (newton-meters)
-0.060
-0.062
-0.064
-0.066
-0.068
-0.070
-0.072
-0.074
18600
18700
18800
18900
19000
19100
19200
Time (sec)
--- 2-axis control, standard method
--- 2-axis control, recursive method

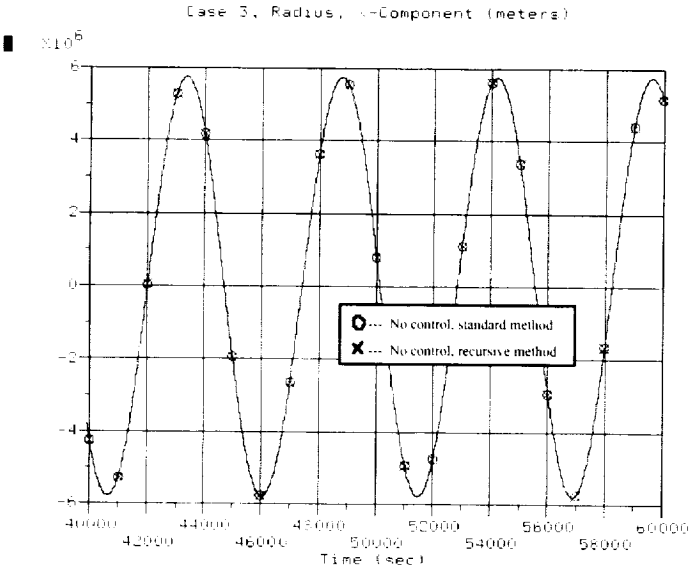

Case 2, Aerodynamic Torque, z-Component (newton-meters)
-0.066
-0.068
-0.070
-0.072
-0.074
-0.076
-0.078
-0.080
-0.082
52500
52600
52700
52800
52900
53000
53100
53200
53300
53400
53500
Time (sec)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method

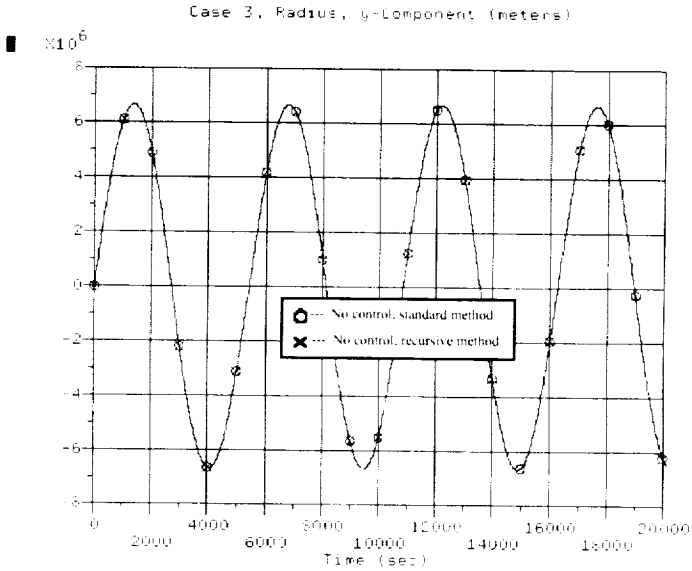

Case 2, Aerodynamic Torque, z-Component (newton-meters)
-0.048
-0.049
-0.050
-0.051
-0.052
-0.053
-0.054
-0.055
-0.056
-0.057
-0.058
59500
59550
59600
59650
59700
59750
59800
59850
59900
59950
60000
Time (sec)
O --- 2-axis control, standard method
X --- 2-axis control, recursive method

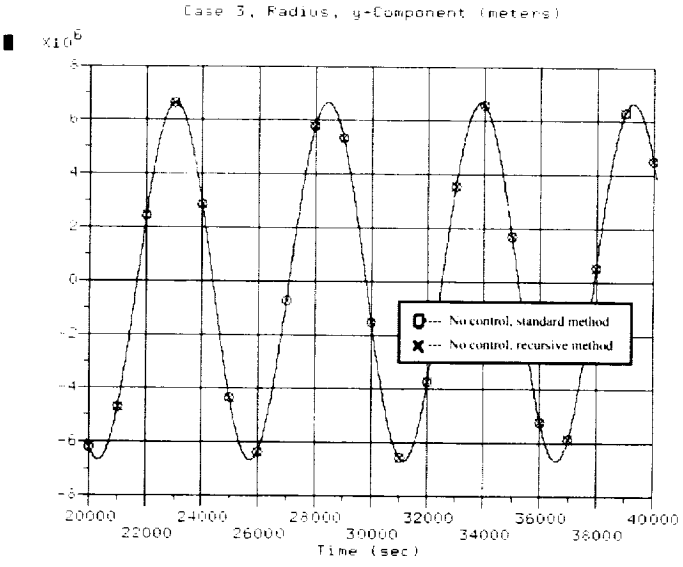
A.3 Response Plots for Case 3 (No Attitude Control)

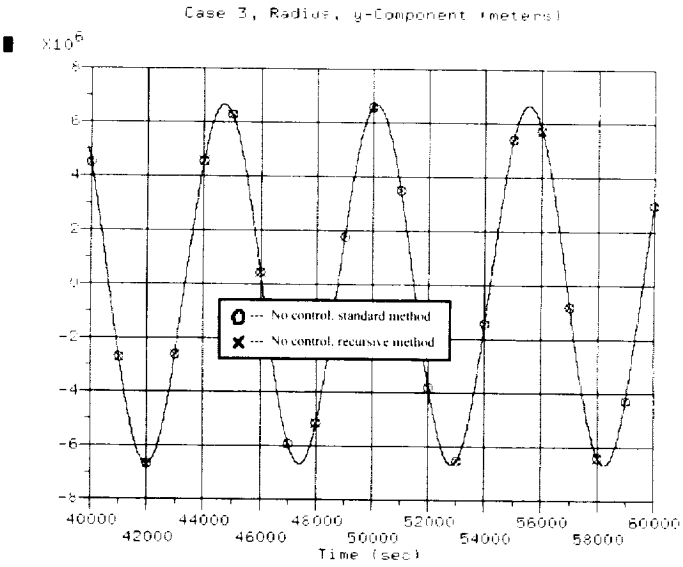

Case 3, Radius, x-Component (meters)
X10^6
No control, standard method
No control, recursive method
Time (sec)


Case 3, Radius, x-Component (meters)
X10^6
No control, standard method
No control, recursive method
Time (sec)

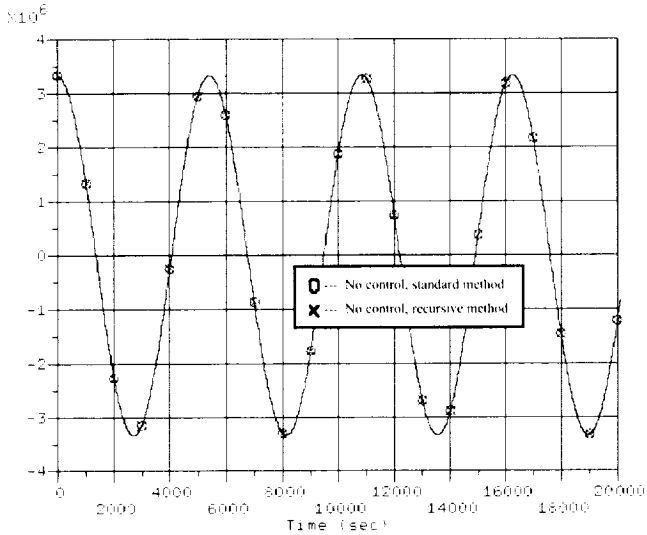

Case 3, Radius, x-Component (meters)
X10^6
No control, standard method
No control, recursive method
Time (sec)


Case 3, Radius, y-Component (meters)
X10^6
No control, standard method
No control, recursive method
Time (sec)

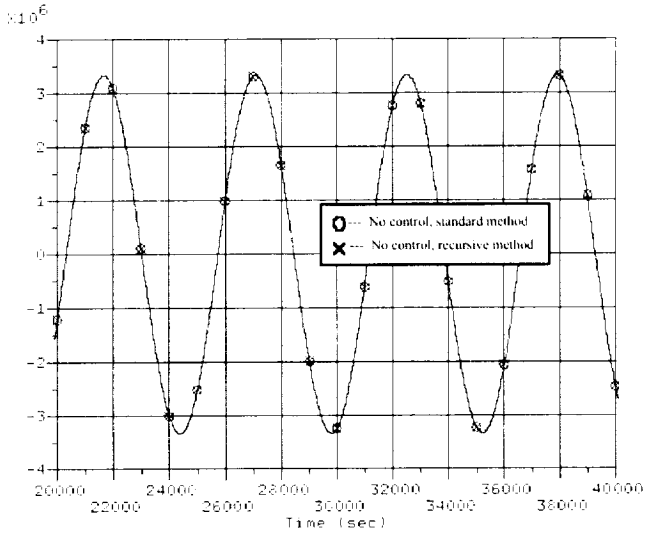

Case 3, Radius, y-Component (meters)
X10^6
No control, standard method
No control, recursive method
Time (sec)


Case 3, Radius, y-Component (meters)
X10^6
No control, standard method
No control, recursive method
Time (sec)

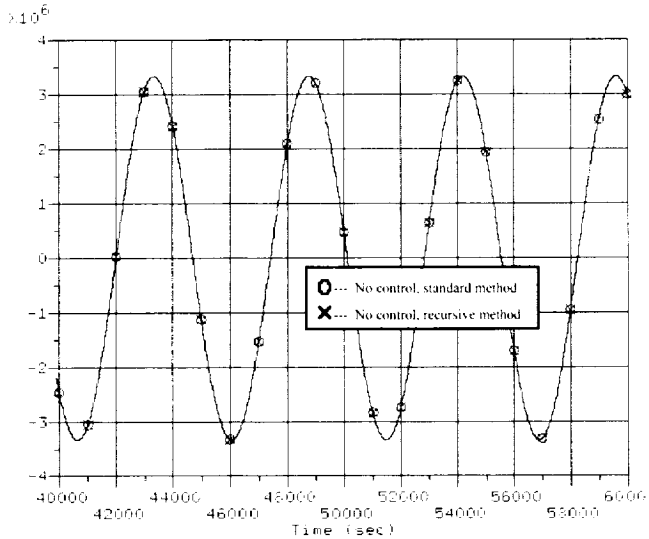
Case 3, Radius, z-Component (meters)



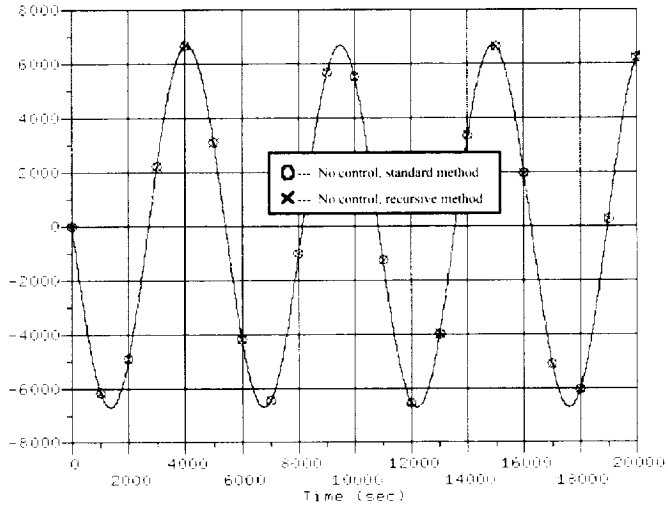
Case 3, Radius, z-Component (meters)



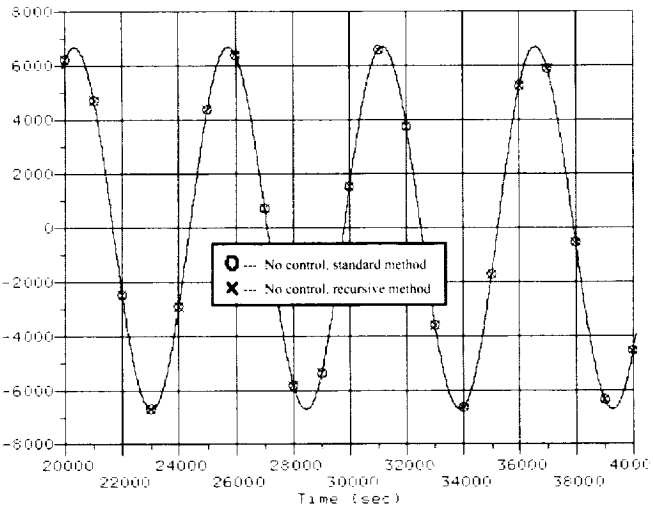
Case 3, Radius, z-Component (meters)



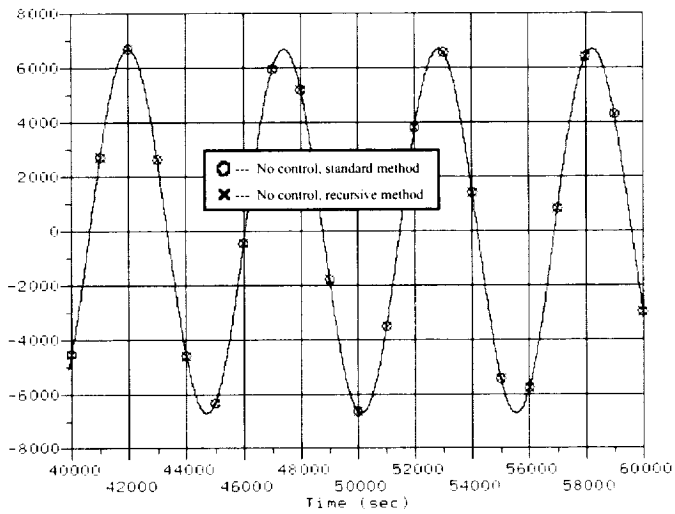
Case 3, Velocity, x-Component (meters/sec)

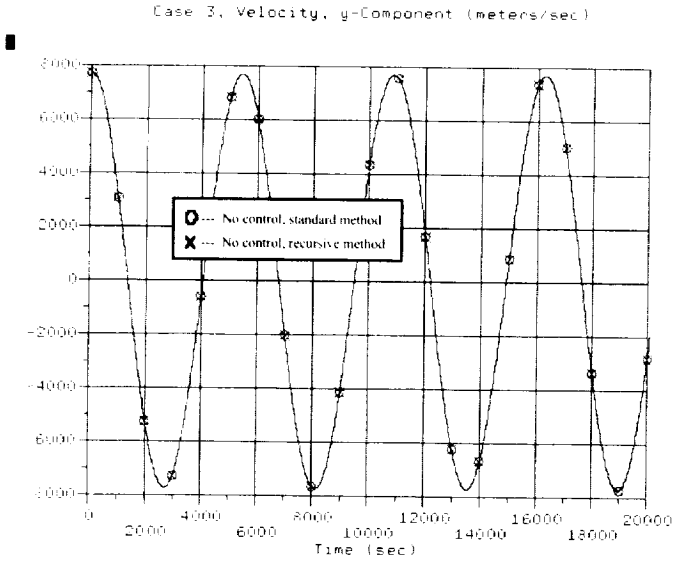


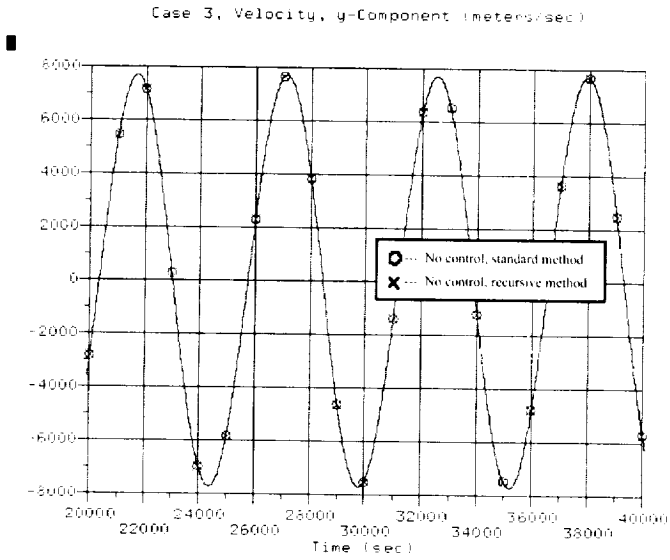
Case 3, Velocity, x-Component (meters/sec)

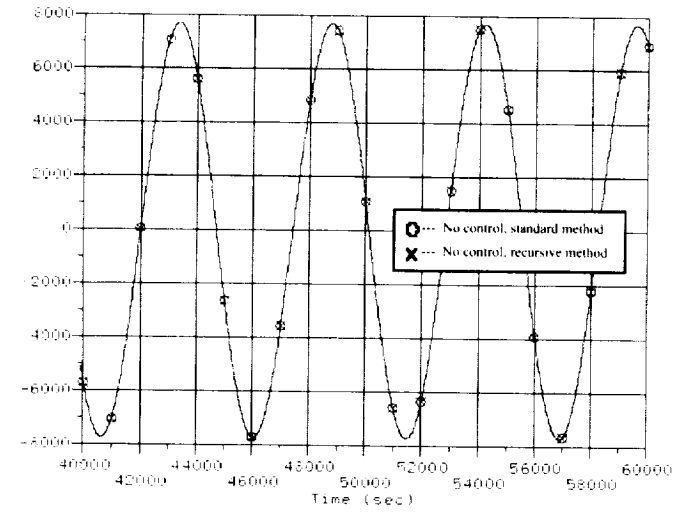


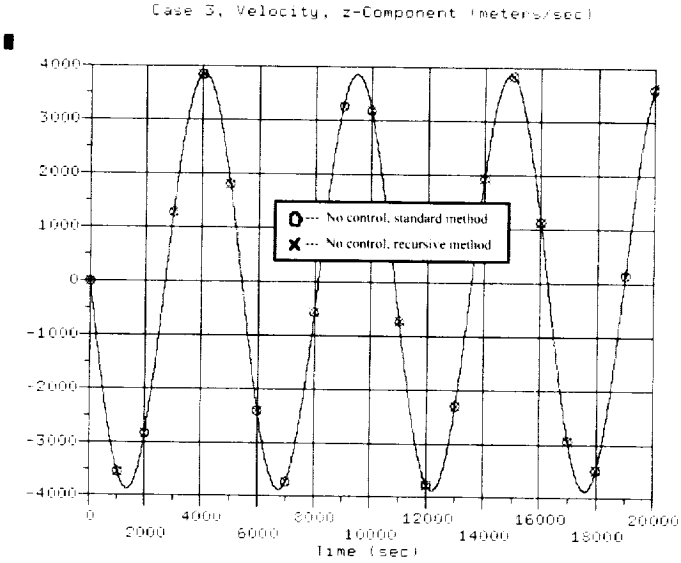
Case 3, Velocity, x-Component (meters/sec)

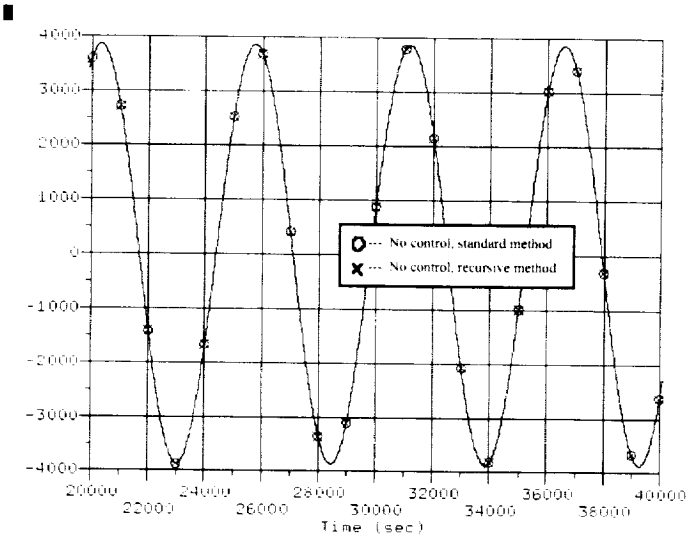


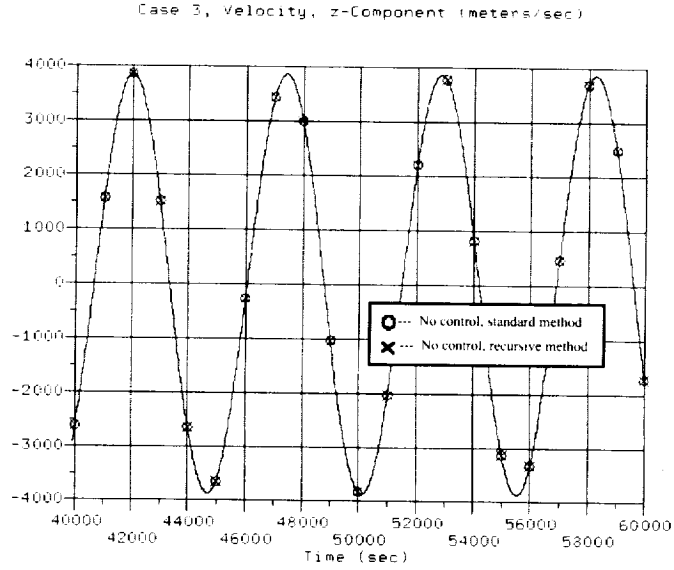

Case 3, Velocity, y-Component (meters/sec)
O --- No control, standard method
X --- No control, recursive method
Time (sec)

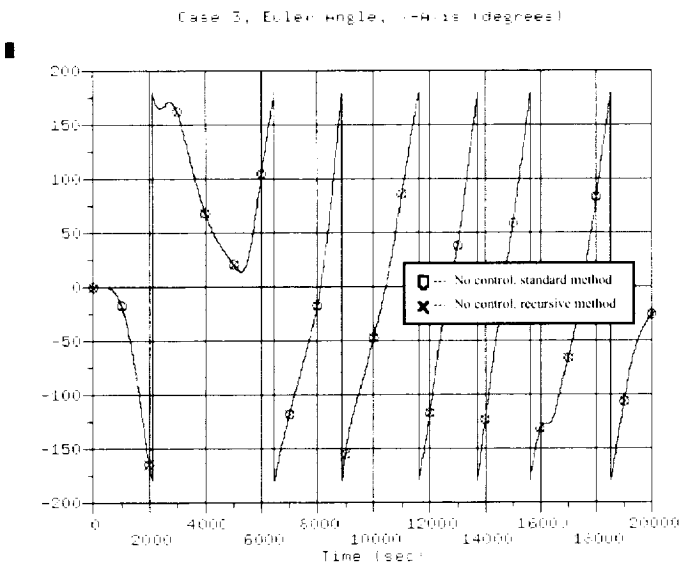

Case 3, Velocity, y-Component (meters/sec)
O --- No control, standard method
X --- No control, recursive method
Time (sec)

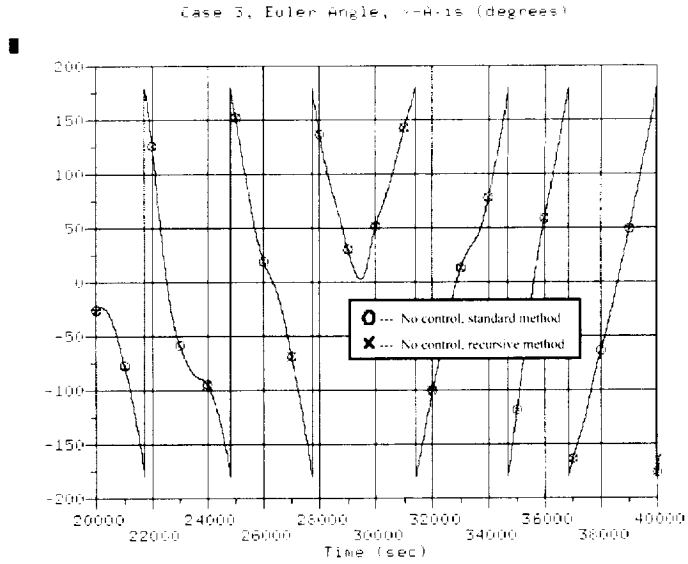

Case 3, Velocity, y-Component (meters/sec)
O --- No control, standard method
X --- No control, recursive method
Time (sec)

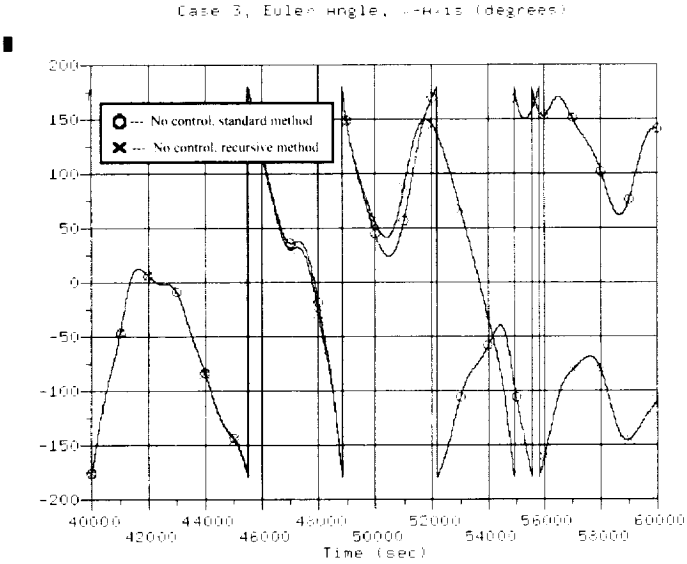

Case 3, Velocity, z-Component (meters/sec)
O --- No control, standard method
X --- No control, recursive method
Time (sec)

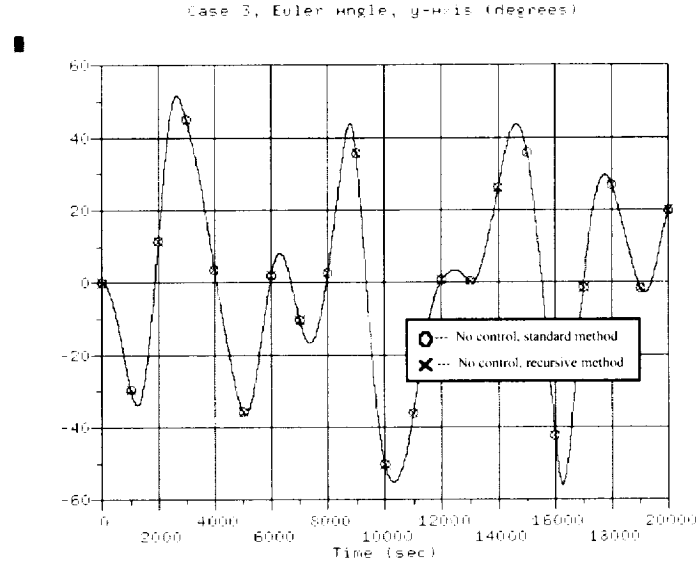

Case 3, Velocity, z-Component (meters/sec)
O --- No control, standard method
X --- No control, recursive method
Time (sec)

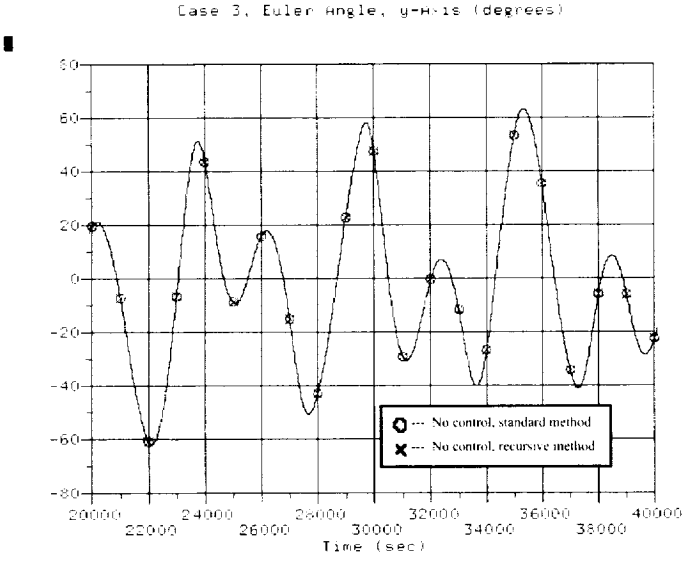

Case 3, Velocity, z-Component (meters/sec)
O --- No control, standard method
X --- No control, recursive method
Time (sec)

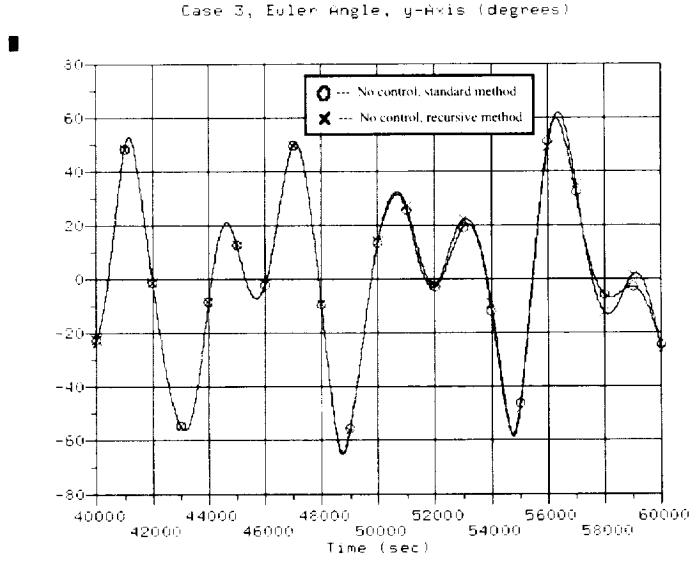

Case 3, Euler Angle, x-Axis (degrees)
No control, standard method
No control, recursive method
Time (sec)

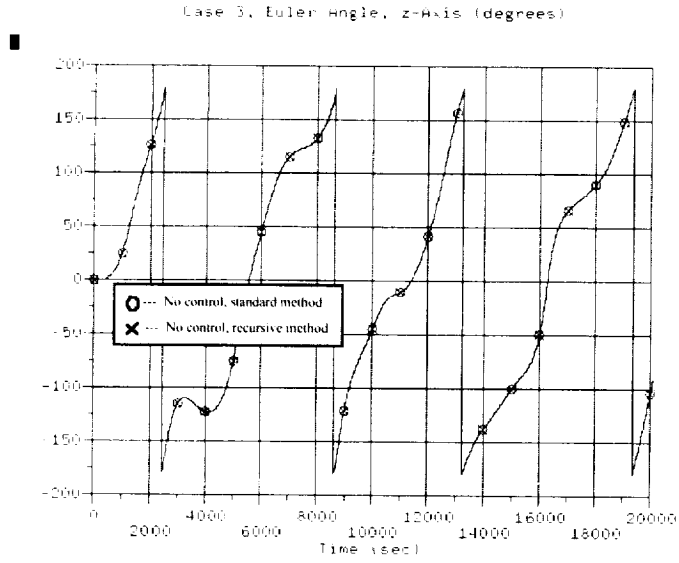

Case 3, Euler Angle, x-Axis (degrees)
No control, standard method
No control, recursive method
Time (sec)

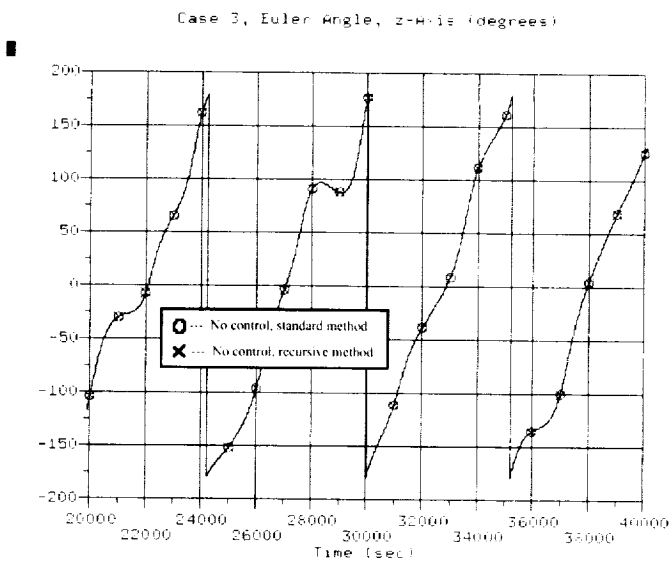

Case 3, Euler Angle, x-Axis (degrees)
No control, standard method
No control, recursive method
Time (sec)

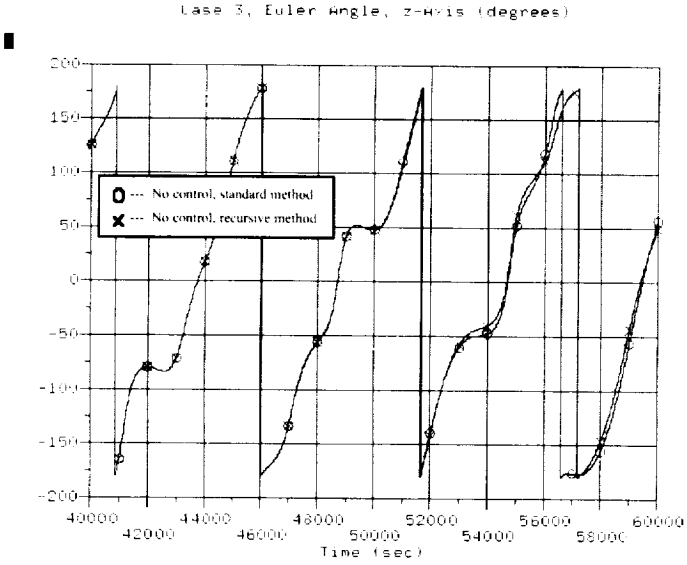

Case 3, Euler Angle, y-Axis (degrees)
No control, standard method
No control, recursive method
Time (sec)

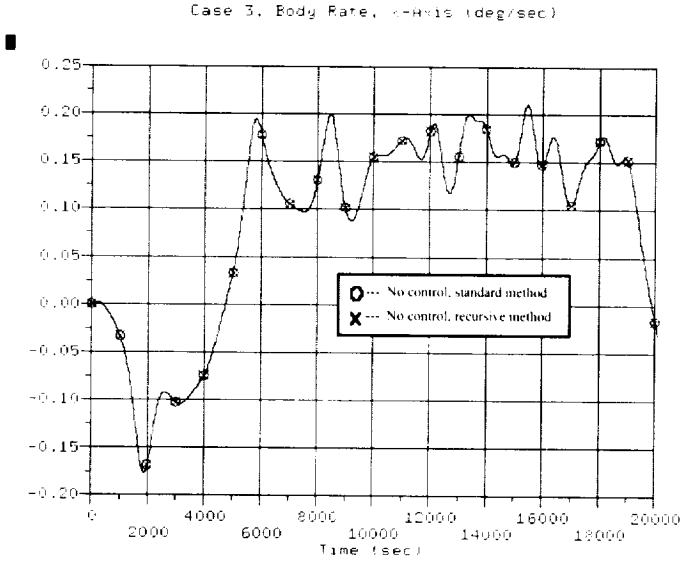

Case 3, Euler Angle, y-Axis (degrees)
No control, standard method
No control, recursive method
Time (sec)

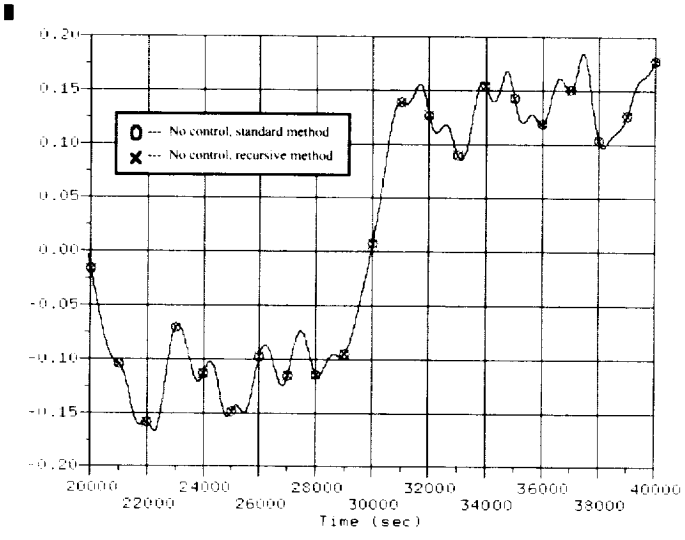

Case 3, Euler Angle, y-Axis (degrees)
No control, standard method
No control, recursive method
Time (sec)

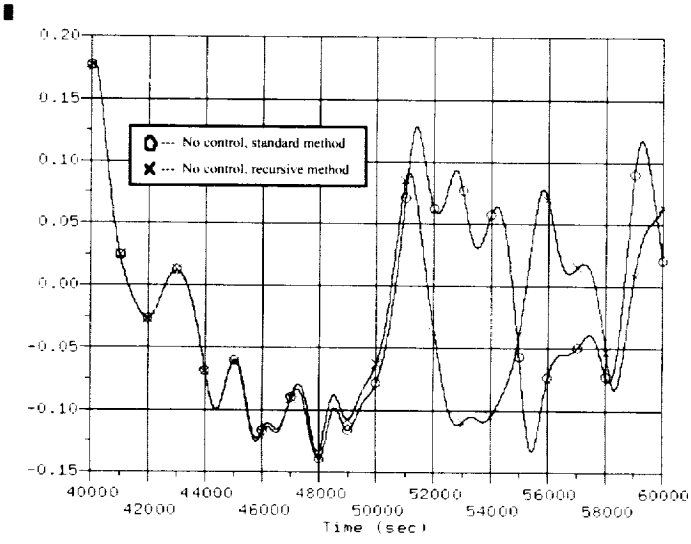

Case 3, Euler Angle, z-Axis (degrees)
O --- No control, standard method
X --- No control, recursive method
Time (sec)

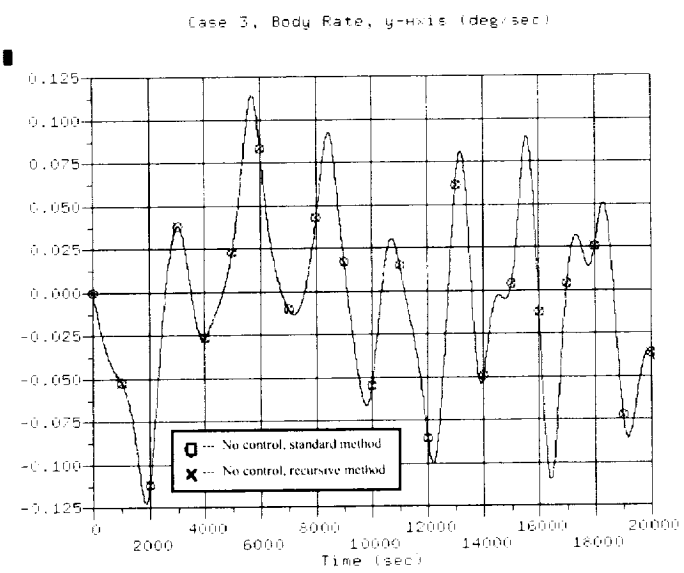

Case 3, Euler Angle, z-Axis (degrees)
O --- No control, standard method
X --- No control, recursive method
Time (sec)

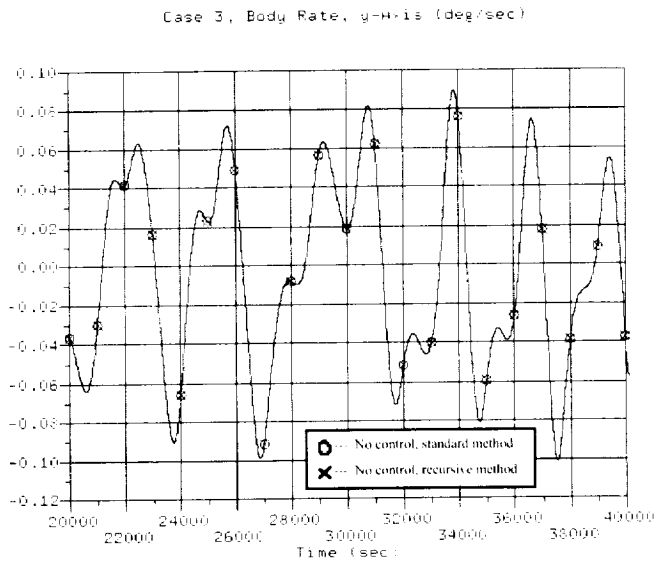

Case 3, Euler Angle, z-Axis (degrees)
O --- No control, standard method
X --- No control, recursive method
Time (sec)

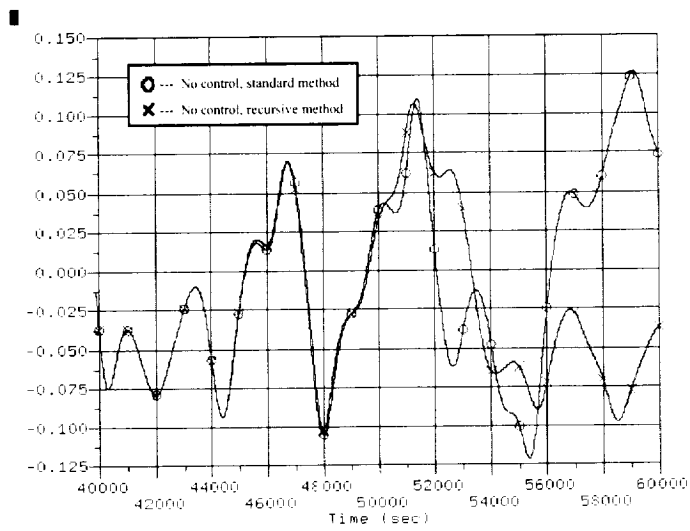

Case 3, Body Rate, x-Axis (deg/sec)
O --- No control, standard method
X --- No control, recursive method
Time (sec)

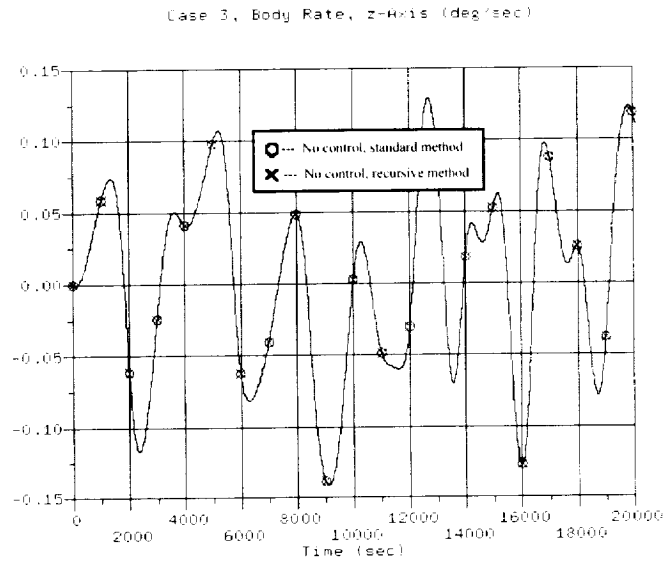

Case 3, Body Rate, x-Axis (deg/sec)
O --- No control, standard method
X --- No control, recursive method
Time (sec)

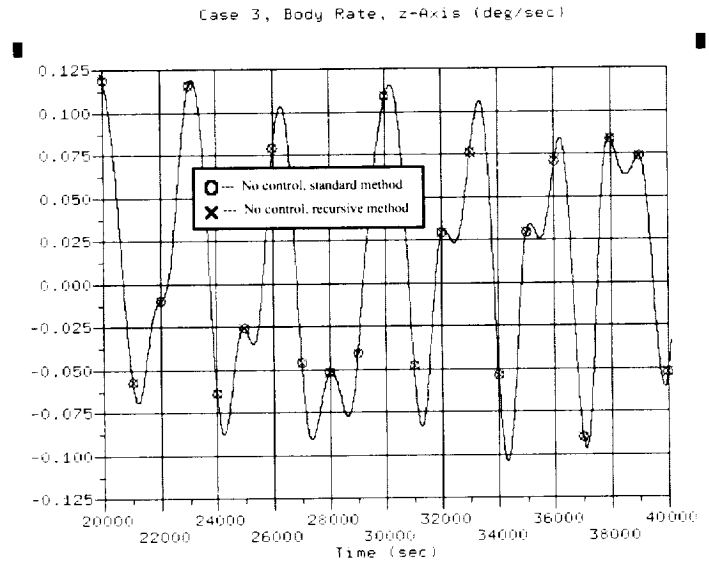

Case 3, Body Rate, x-Axis (deg/sec)
O --- No control, standard method
X --- No control, recursive method
Time (sec)

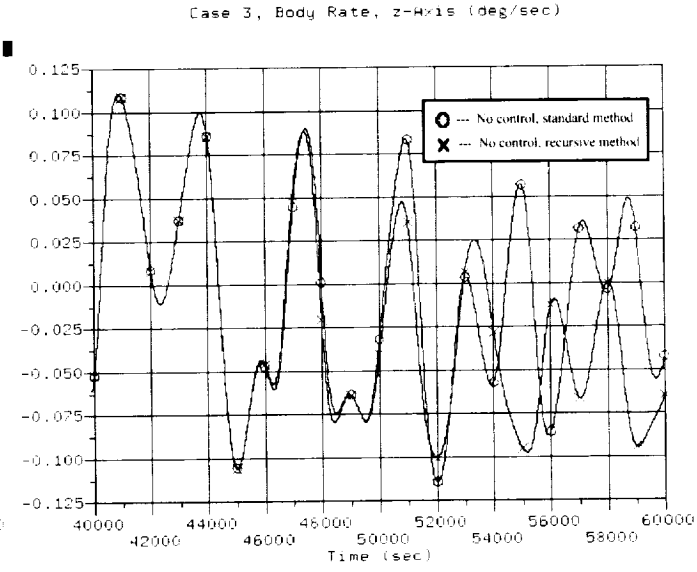

Case 3, Body Rate, y-Axis (deg/sec)
0.125
0.100
0.075
0.050
0.025
0.000
-0.025
-0.050
-0.075
-0.100
-0.125
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Time (sec)
--- No control, standard method
--- No control, recursive method


Case 3, Body Rate, y-Axis (deg/sec)
0.10
0.08
0.06
0.04
0.02
0.00
-0.02
-0.04
-0.06
-0.08
-0.10
-0.12
20000
22000
24000
26000
28000
30000
32000
34000
36000
38000
40000
Time (sec)
--- No control, standard method
--- No control, recursive method

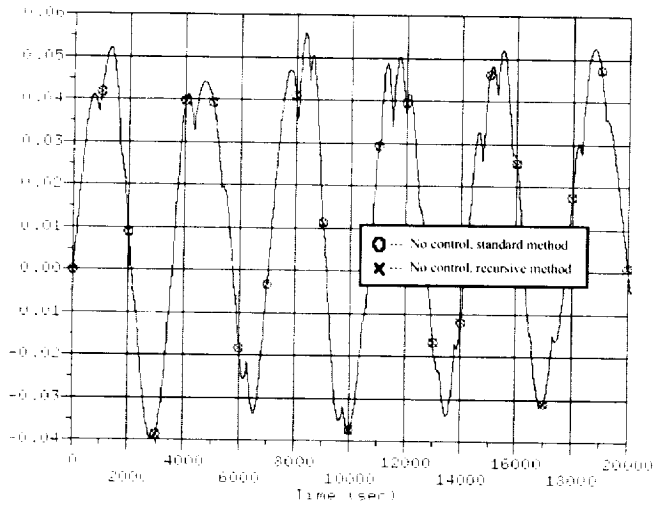

Case 3, Body Rate, y-Axis (deg/sec)
0.150
0.125
0.100
0.075
0.050
0.025
0.000
-0.025
-0.050
-0.075
-0.100
-0.125
40000
42000
44000
46000
48000
50000
52000
54000
56000
58000
60000
Time (sec)
--- No control, standard method
--- No control, recursive method


Case 3, Body Rate, z-Axis (deg/sec)
0.15
0.10
0.05
0.00
-0.05
-0.10
-0.15
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Time (sec)
--- No control, standard method
--- No control, recursive method

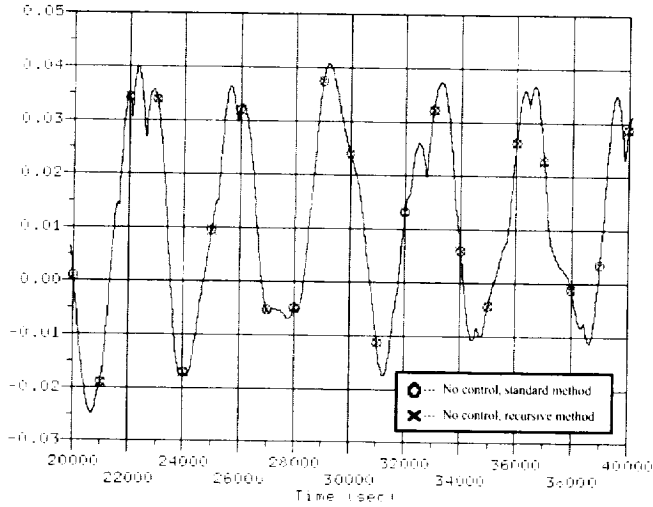

Case 3, Body Rate, z-Axis (deg/sec)
0.125
0.100
0.075
0.050
0.025
0.000
-0.025
-0.050
-0.075
-0.100
-0.125
20000
22000
24000
26000
28000
30000
32000
34000
36000
38000
40000
Time (sec)
--- No control, standard method
--- No control, recursive method


Case 3, Body Rate, z-Axis (deg/sec)
0.125
0.100
0.075
0.050
0.025
0.000
-0.025
-0.050
-0.075
-0.100
-0.125
40000
42000
44000
46000
48000
50000
52000
54000
56000
58000
60000
Time (sec)
--- No control, standard method
--- No control, recursive method

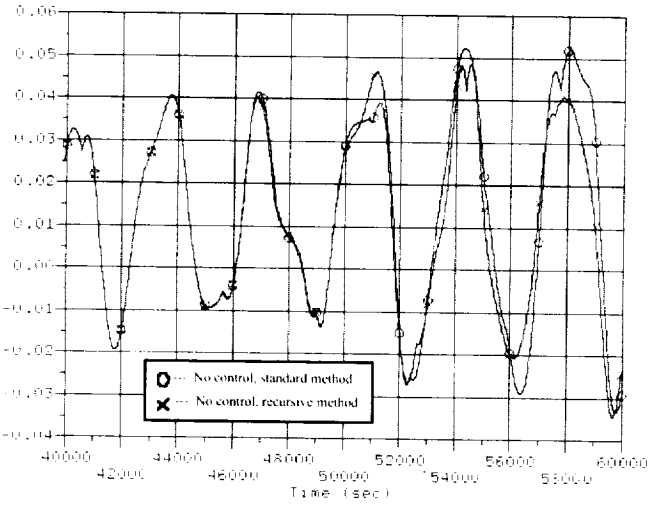
Case 3, Aerodynamic Force, x-Component (newtons)



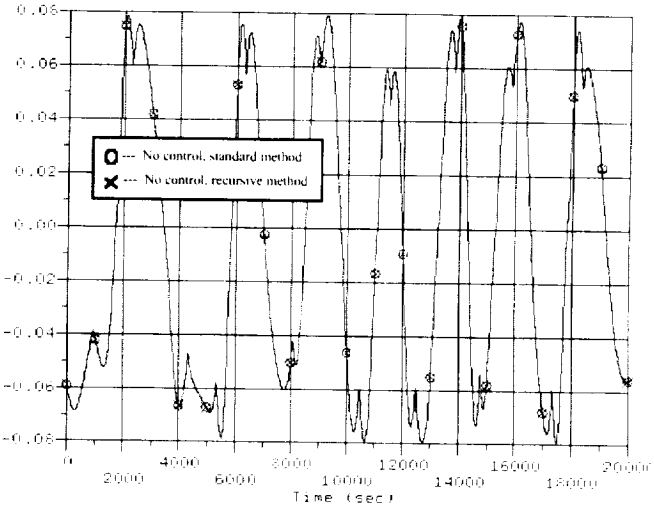
Case 3, Aerodynamic Force, x-Component (newtons)



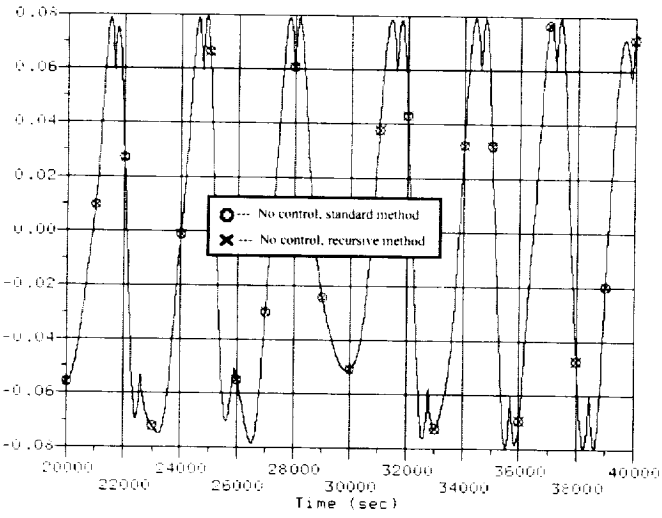
Case 3, Aerodynamic Force, x-Component (newtons)



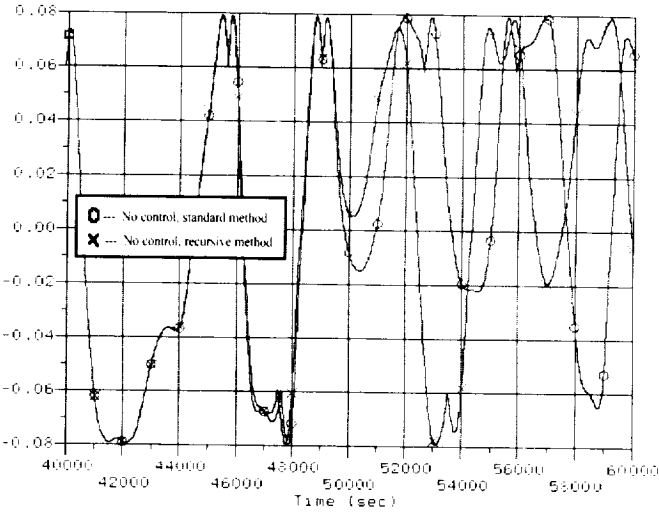
Case 3, Aerodynamic Force, y-Component (newtons)



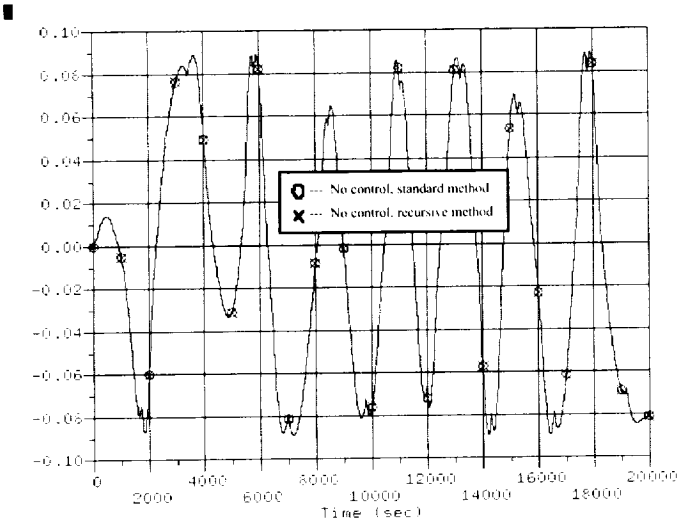
Case 3, Aerodynamic Force, y-Component (newtons)



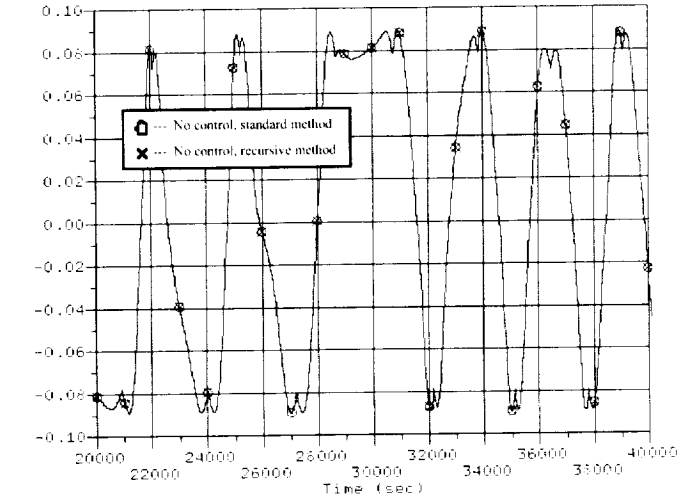
Case 3, Aerodynamic Force, y-Component (newtons)



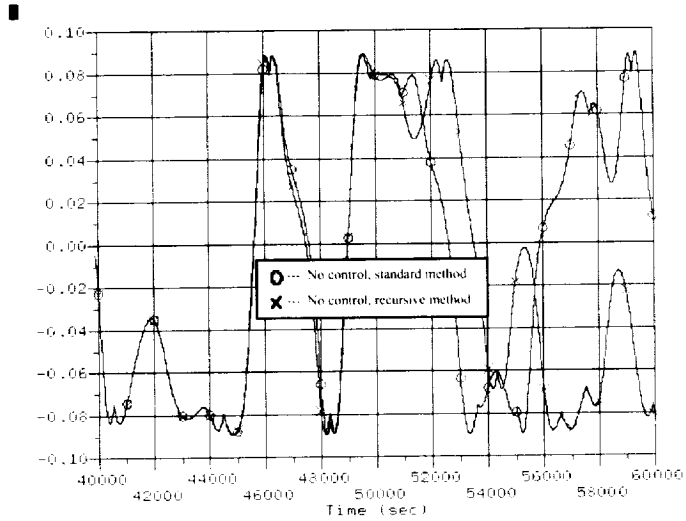
Case 3, Aerodynamic Force, z-Component (newtons)



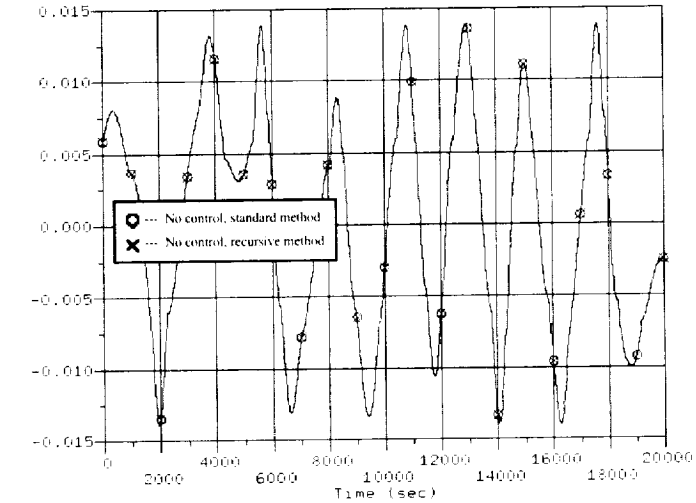
Case 3, Aerodynamic Force, z-Component (newtons)



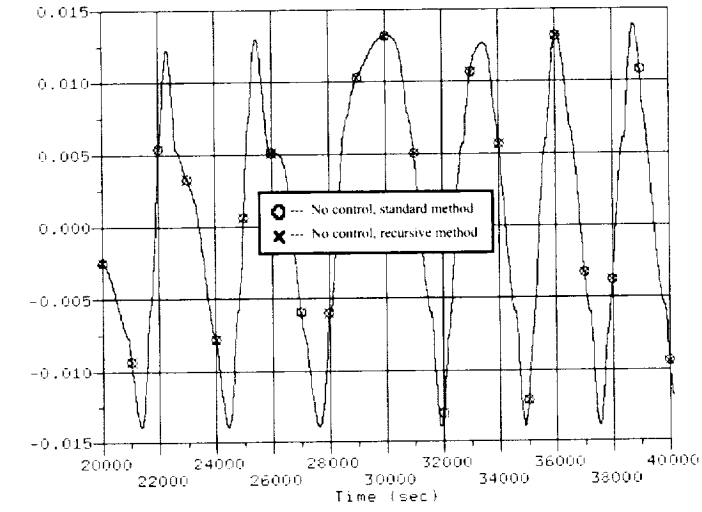
Case 3, Aerodynamic Force, z-Component (newtons)



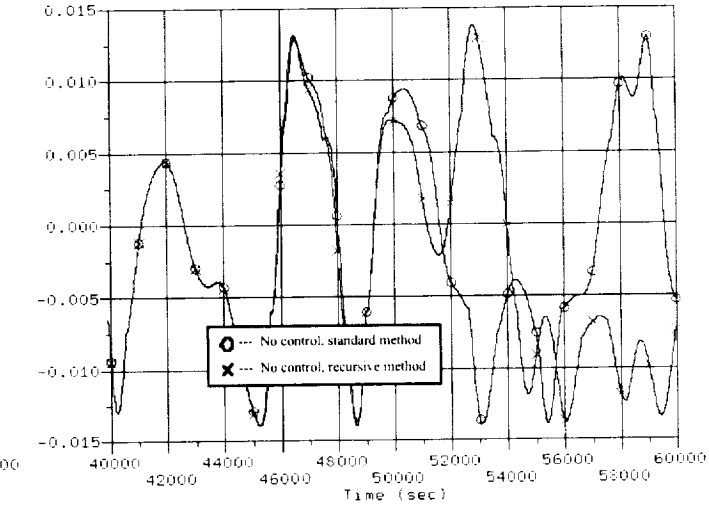
Case 3, Aerodynamic Torque, x-Component (newton-meters)



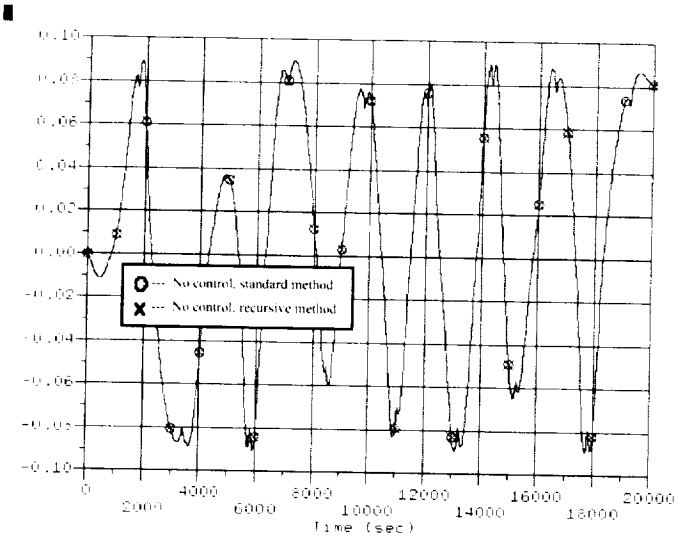
Case 3, Aerodynamic Torque, x-Component (newton-meters)



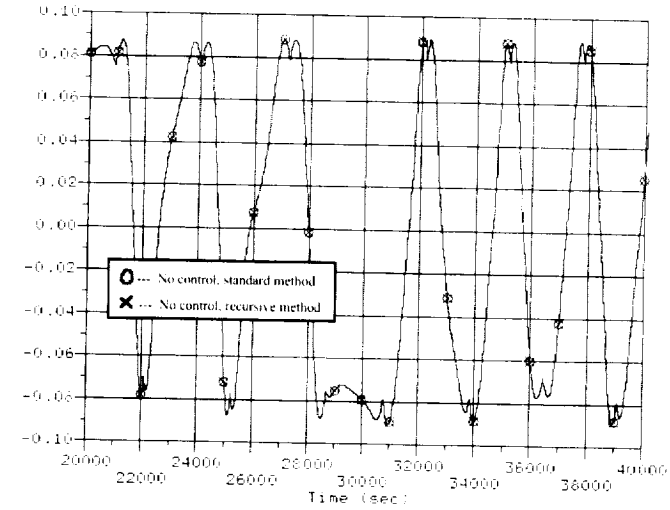
Case 3, Aerodynamic Torque, x-Component (newton-meters)



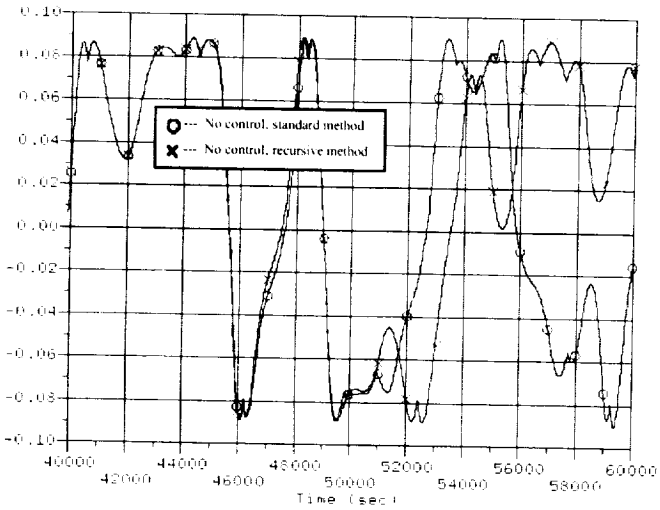
Case 3, Aerodynamic Torque, y-Component (newton-meters)



Case 3, Aerodynamic Torque, y-Component (newton-meters)

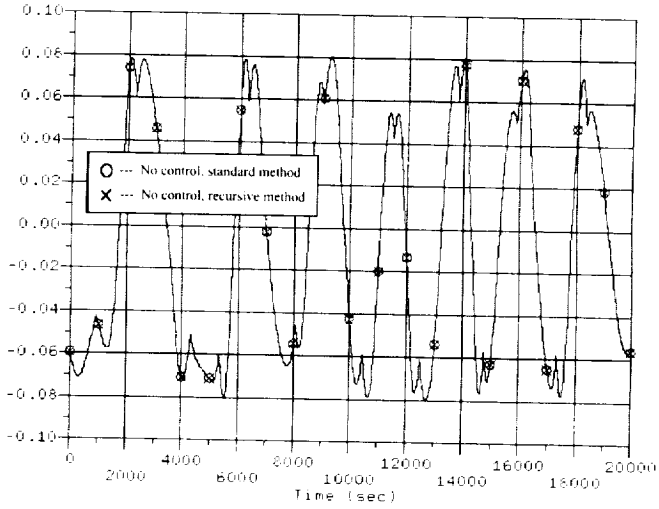


Case 3, Aerodynamic Torque, y-Component (newton-meters)

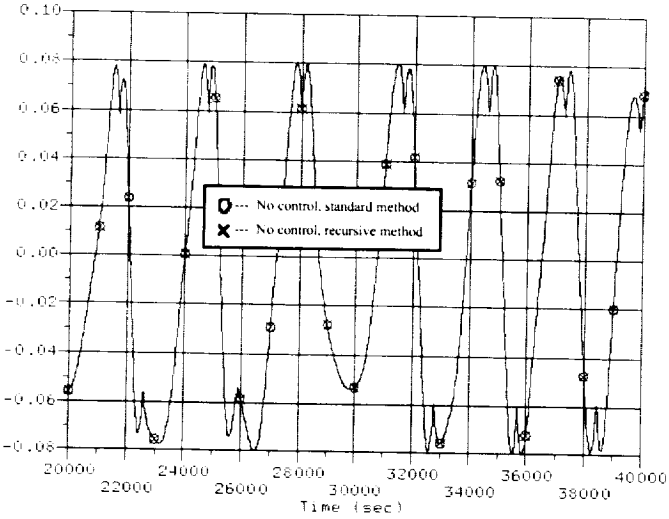


Case 3, Aerodynamic Torque, z-Component (newton-meters)

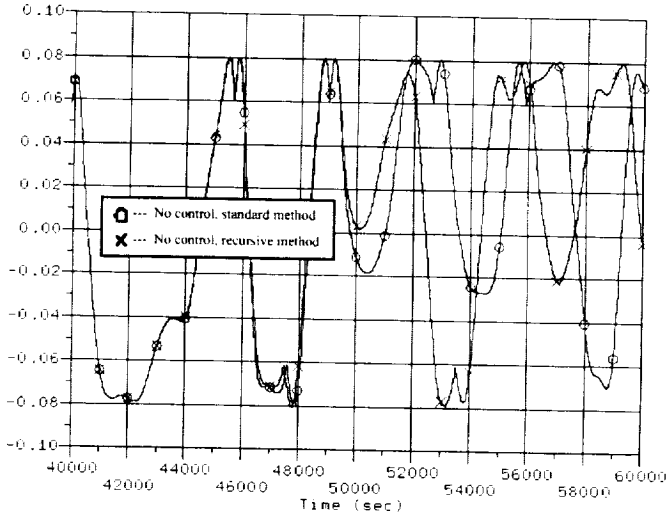
1-1 ⟷ 19-1 1-2 ⟷ 19-2

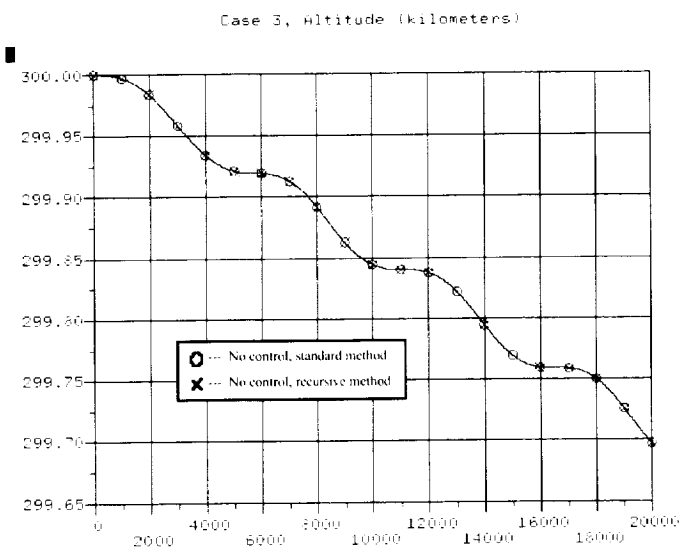


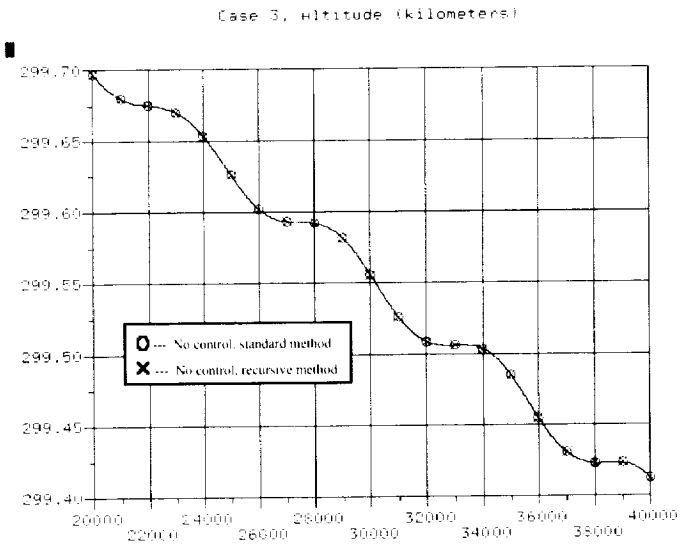
Case 3, Aerodynamic Torque, z-Component (newton-meters)

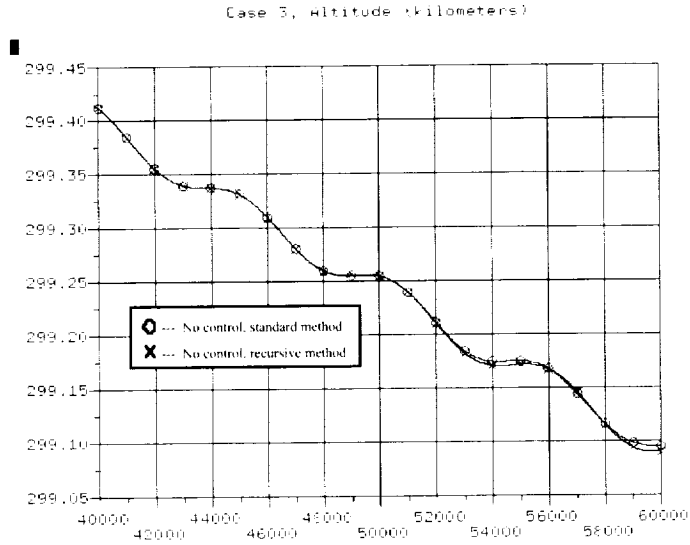


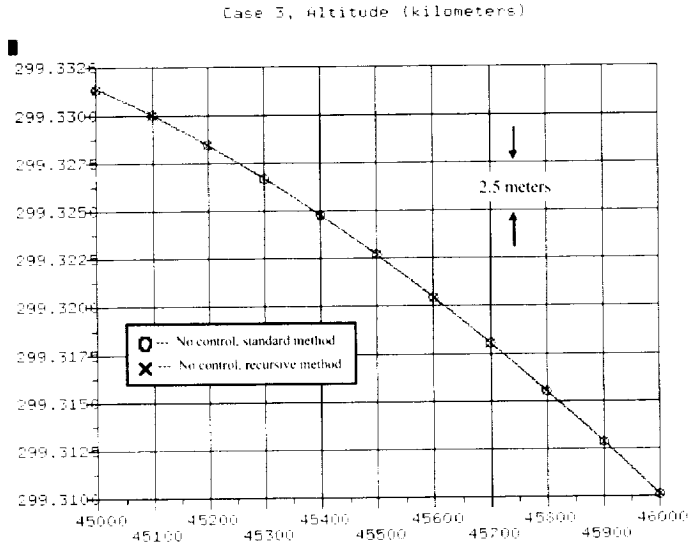
Case 3, Aerodynamic Torque, z-Component (newton-meters)

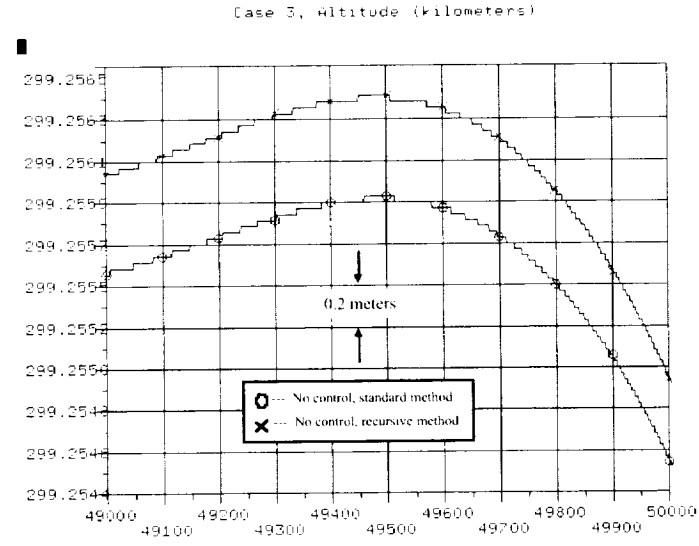


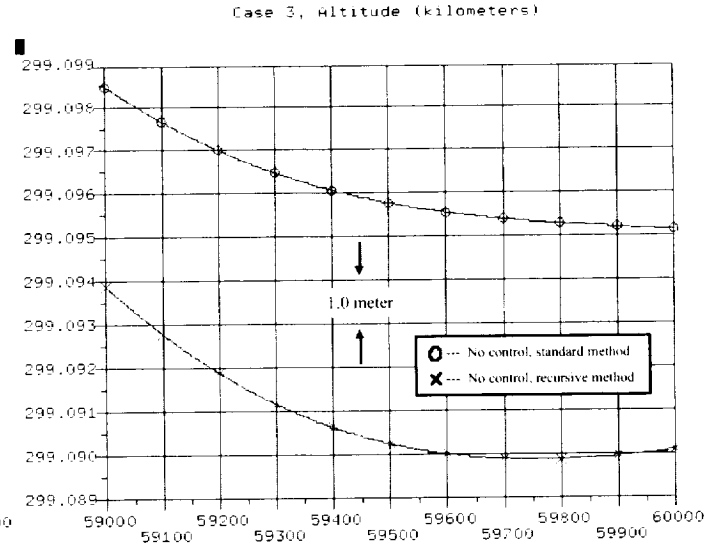

Case 3, Altitude (kilometers)
300.00
299.95
299.90
299.85
299.80
299.75
299.70
299.65
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
O --- No control, standard method
X --- No control, recursive method


Case 3, Altitude (kilometers)
299.70
299.65
299.60
299.55
299.50
299.45
299.40
20000
22000
24000
26000
28000
30000
32000
34000
36000
38000
40000
O --- No control, standard method
X --- No control, recursive method

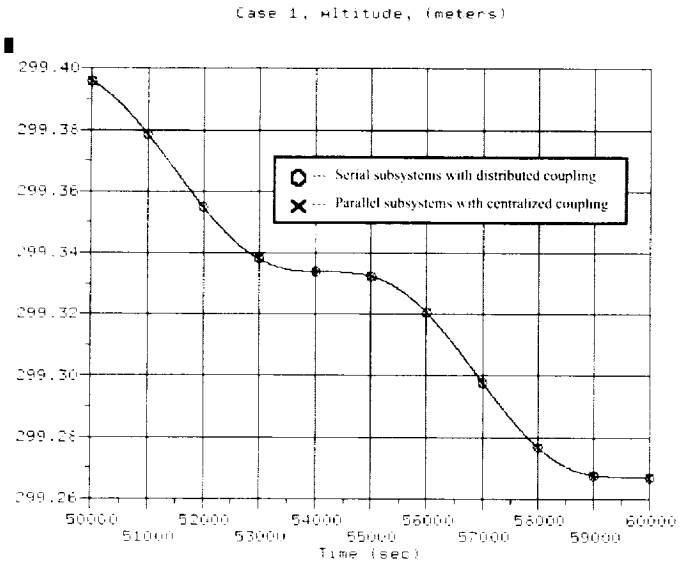

Case 3, Altitude (kilometers)
299.45
299.40
299.35
299.30
299.25
299.20
299.15
299.10
299.05
40000
42000
44000
46000
48000
50000
52000
54000
56000
58000
60000
O --- No control, standard method
X --- No control, recursive method

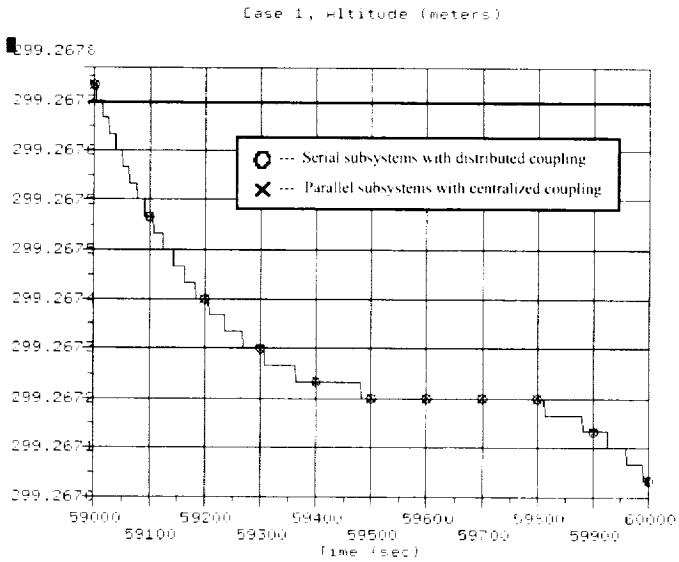

Case 3, Altitude (kilometers)
299.3325
299.3300
299.3275
299.3250
299.3225
299.3200
299.3175
299.3150
299.3125
299.3100
45000
45100
45200
45300
45400
45500
45600
45700
45800
45900
46000
2.5 meters
O --- No control, standard method
X --- No control, recursive method

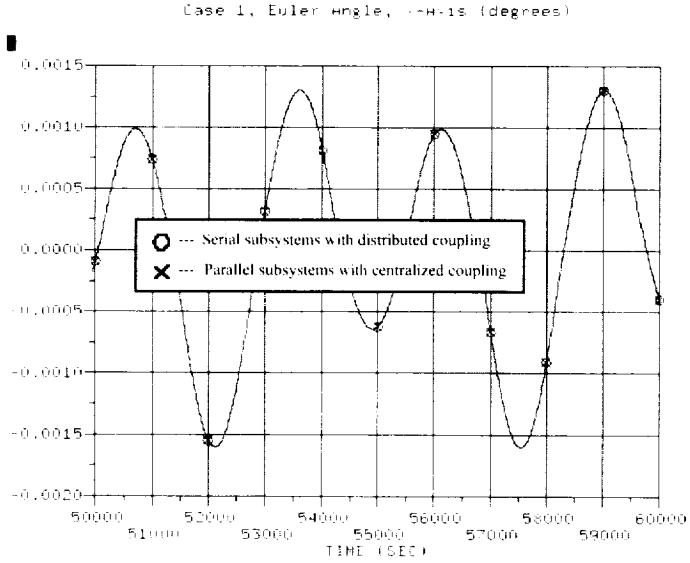

Case 3, Altitude (kilometers)
299.2565
299.2560
299.2555
299.2550
299.2545
299.2540
49000
49100
49200
49300
49400
49500
49600
49700
49800
49900
50000
0.2 meters
O --- No control, standard method
X --- No control, recursive method

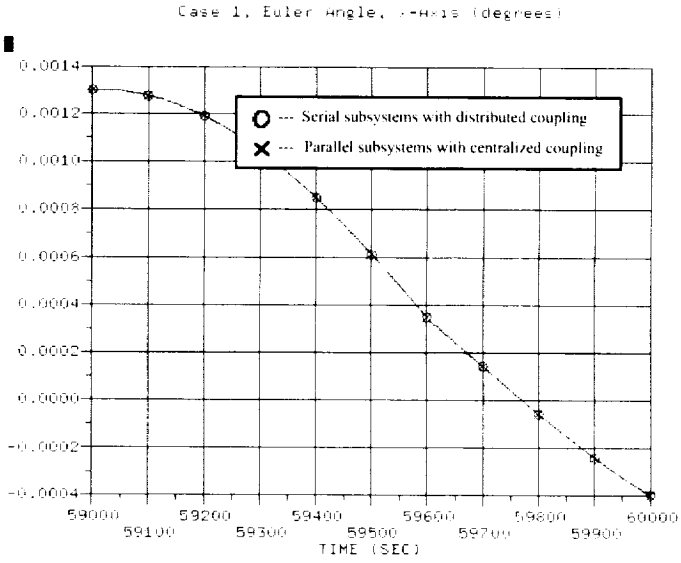

Case 3, Altitude (kilometers)
299.099
299.098
299.097
299.096
299.095
299.094
299.093
299.092
299.091
299.090
299.089
59000
59100
59200
59300
59400
59500
59600
59700
59800
59900
60000
1.0 meter
O --- No control, standard method
X --- No control, recursive method

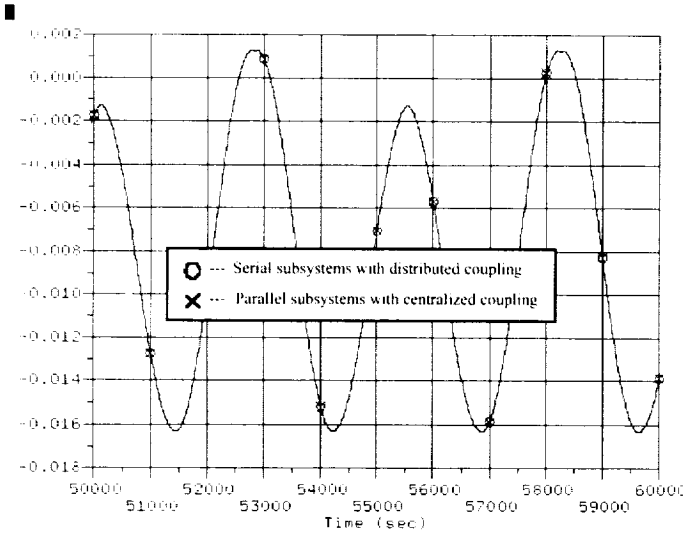
A.4 Response Plots for Cases 1–3 With Central and Distributed Coupling

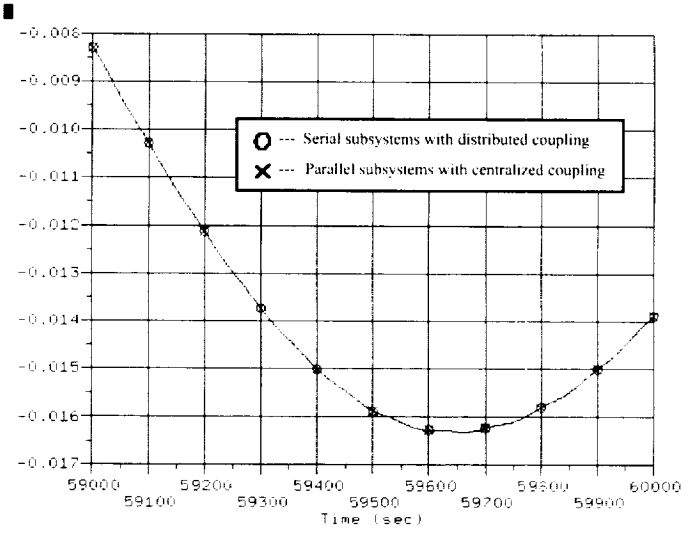

Case 1, Altitude, (meters)
299.40
299.38
299.36
299.34
299.32
299.30
299.28
299.26
50000
51000
52000
53000
54000
55000
56000
57000
58000
59000
60000
Time (sec)
O --- Serial subsystems with distributed coupling
X --- Parallel subsystems with centralized coupling

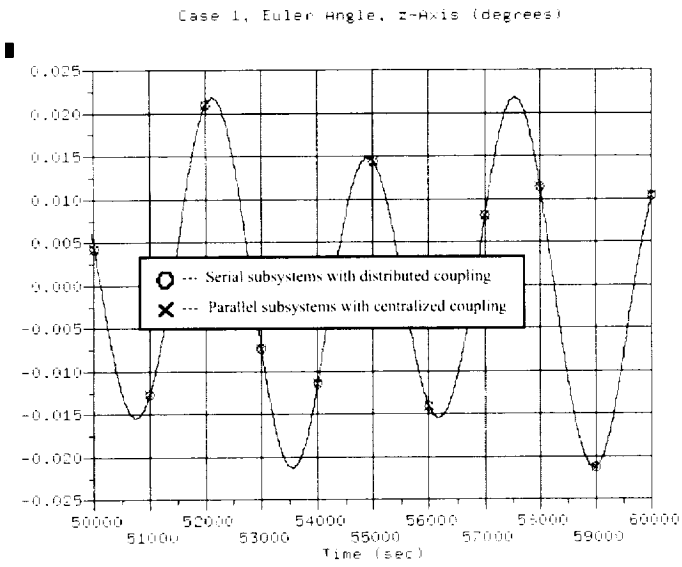

Case 1, Altitude (meters)
299.2678
299.2677
299.2676
299.2675
299.2674
299.2673
299.2672
299.2671
299.2670
59000
59100
59200
59300
59400
59500
59600
59700
59800
59900
60000
Time (sec)
O --- Serial subsystems with distributed coupling
X --- Parallel subsystems with centralized coupling

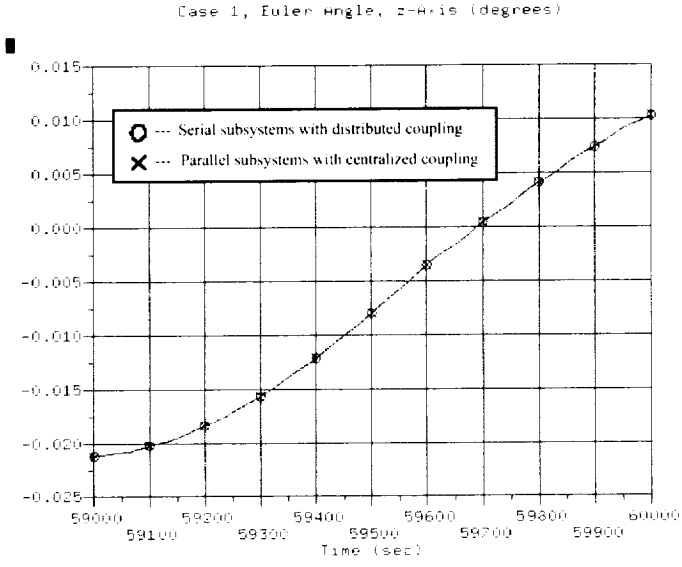

Case 1, Euler Angle, x-Axis (degrees)
0.0015
0.0010
0.0005
0.0000
-0.0005
-0.0010
-0.0015
-0.0020
50000
51000
52000
53000
54000
55000
56000
57000
58000
59000
60000
TIME (SEC)
O --- Serial subsystems with distributed coupling
X --- Parallel subsystems with centralized coupling

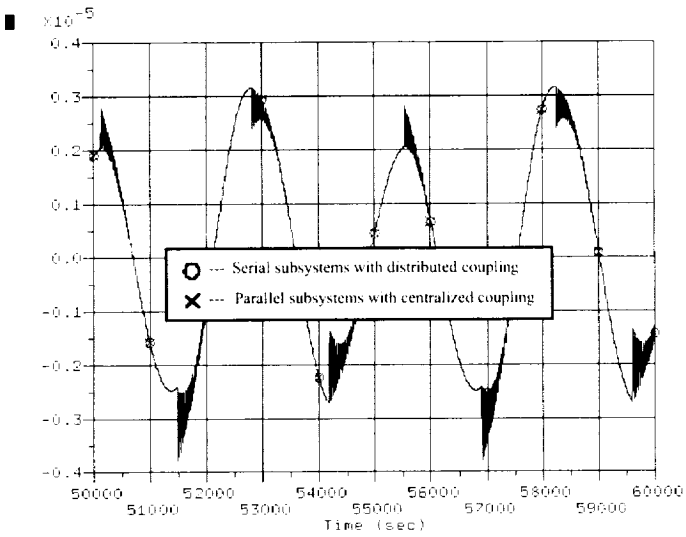

Case 1, Euler Angle, x-Axis (degrees)
0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
-0.0002
-0.0004
59000
59100
59200
59300
59400
59500
59600
59700
59800
59900
60000
TIME (SEC)
O --- Serial subsystems with distributed coupling
X --- Parallel subsystems with centralized coupling

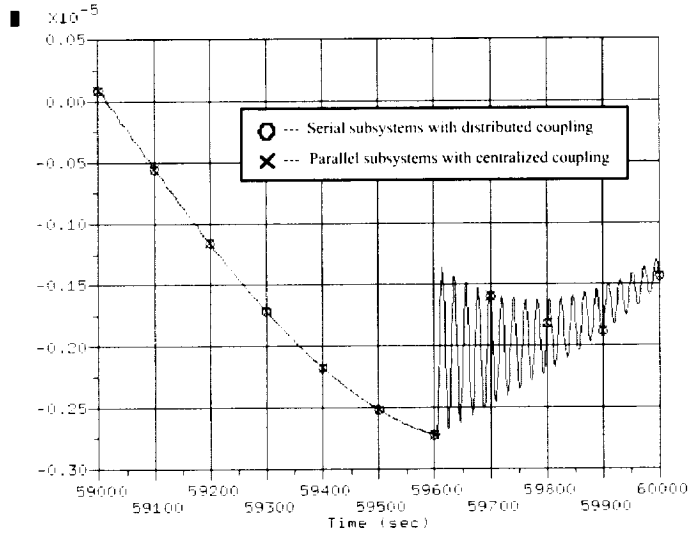

Case 1, Euler Angle, y-Axis (degrees)
0.002
0.000
-0.002
-0.004
-0.006
-0.008
-0.010
-0.012
-0.014
-0.016
-0.018
50000
51000
52000
53000
54000
55000
56000
57000
58000
59000
60000
Time (sec)
O --- Serial subsystems with distributed coupling
X --- Parallel subsystems with centralized coupling

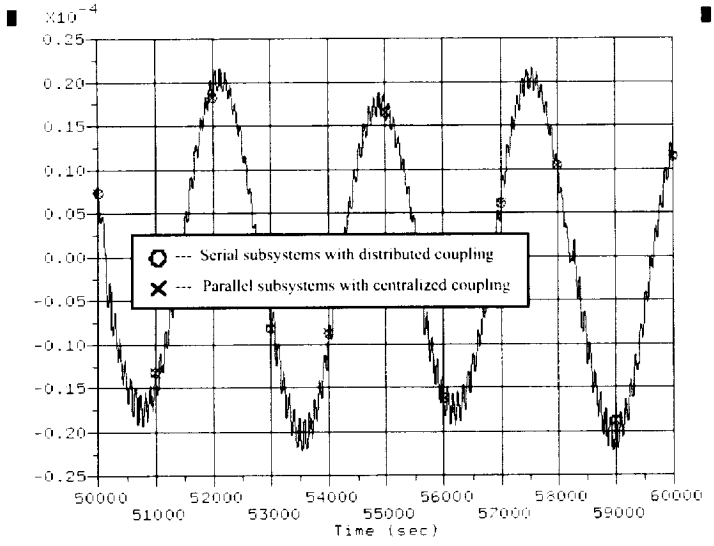

Case 1, Euler Angle, y-Axis (degrees)
-0.008
-0.009
-0.010
-0.011
-0.012
-0.013
-0.014
-0.015
-0.016
-0.017
59000
59100
59200
59300
59400
59500
59600
59700
59800
59900
60000
Time (sec)
O --- Serial subsystems with distributed coupling
X --- Parallel subsystems with centralized coupling

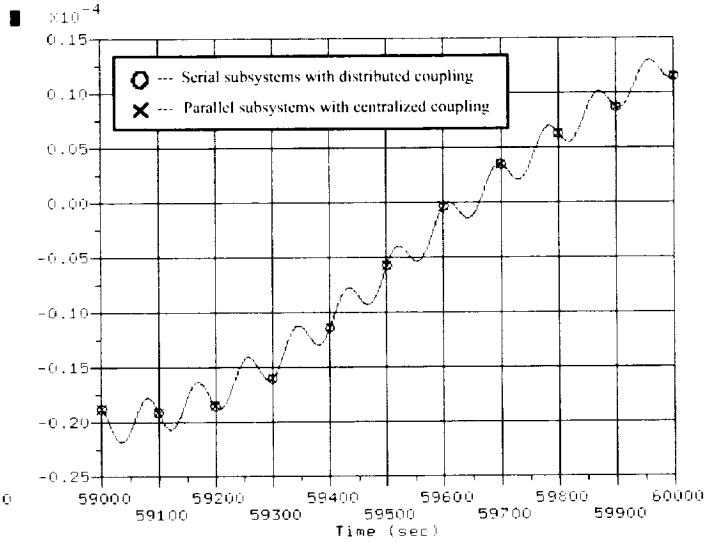

Case 1, Euler Angle, z-Axis (degrees)
0.025
0.020
0.015
0.010
0.005
0.000
-0.005
-0.010
-0.015
-0.020
-0.025
50000
51000
52000
53000
54000
55000
56000
57000
58000
59000
60000
Time (sec)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling

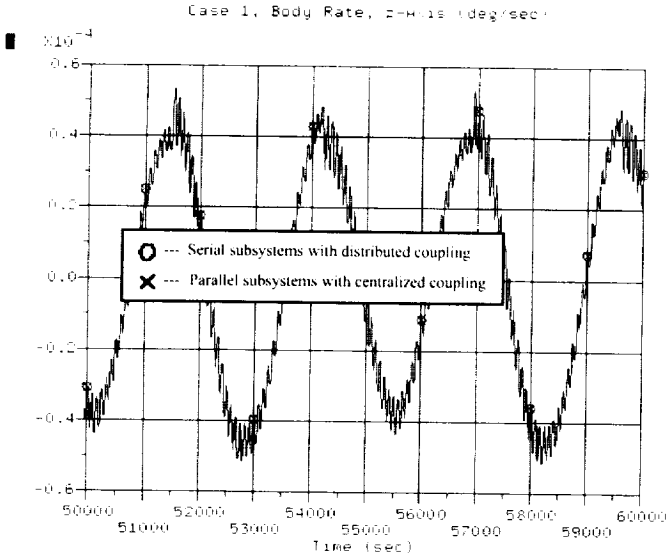

Case 1, Euler Angle, z-Axis (degrees)
0.015
0.010
0.005
0.000
-0.005
-0.010
-0.015
-0.020
-0.025
59000
59100
59200
59300
59400
59500
59600
59700
59800
59900
60000
Time (sec)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling

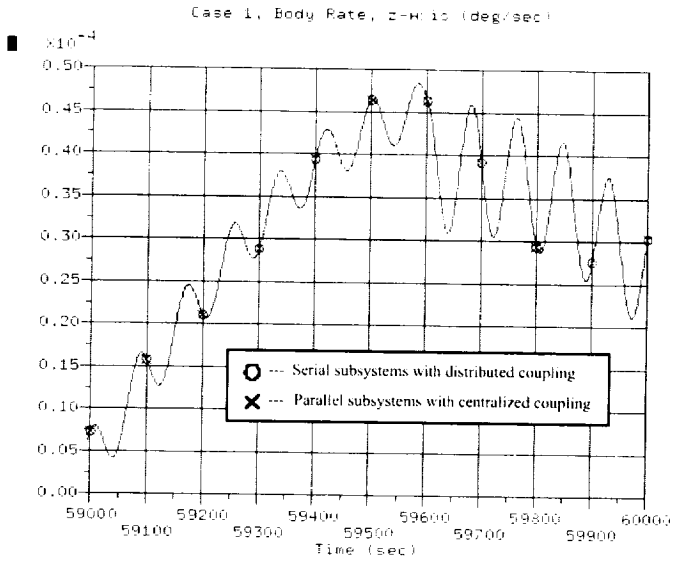

Case 1 Body Rate, x-Axis (deg/sec)
X10^-5
0.4
0.3
0.2
0.1
0.0
-0.1
-0.2
-0.3
-0.4
50000
51000
52000
53000
54000
55000
56000
57000
58000
59000
60000
Time (sec)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling

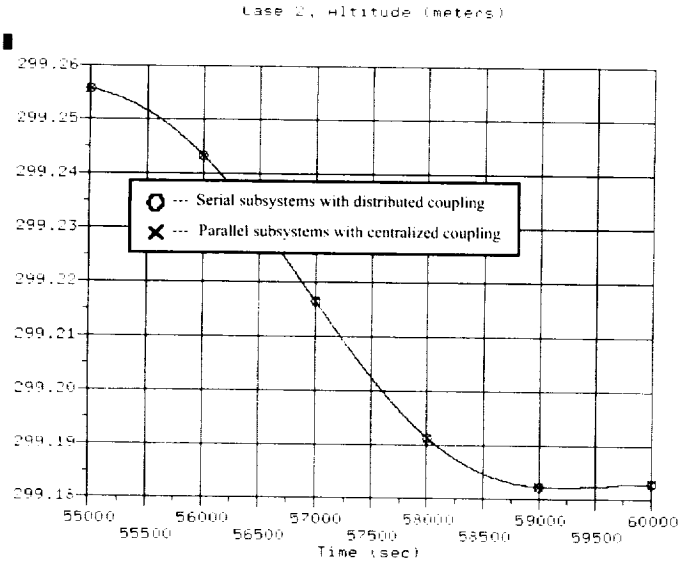

Case 1 Body Rate, x-Axis (deg/sec)
X10^-5
0.05
0.00
-0.05
-0.10
-0.15
-0.20
-0.25
-0.30
59000
59100
59200
59300
59400
59500
59600
59700
59800
59900
60000
Time (sec)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling

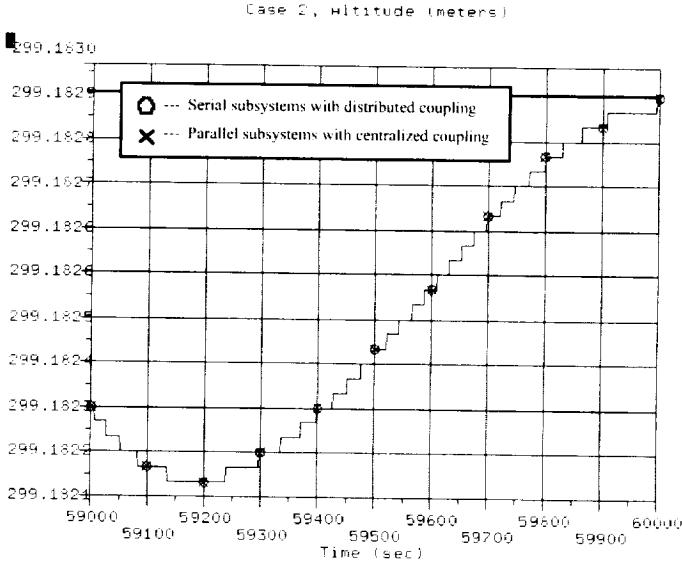

Case 1, Body Rate, y-Axis (deg/sec)
X10^-4
0.25
0.20
0.15
0.10
0.05
0.00
-0.05
-0.10
-0.15
-0.20
-0.25
50000
51000
52000
53000
54000
55000
56000
57000
58000
59000
60000
Time (sec)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling

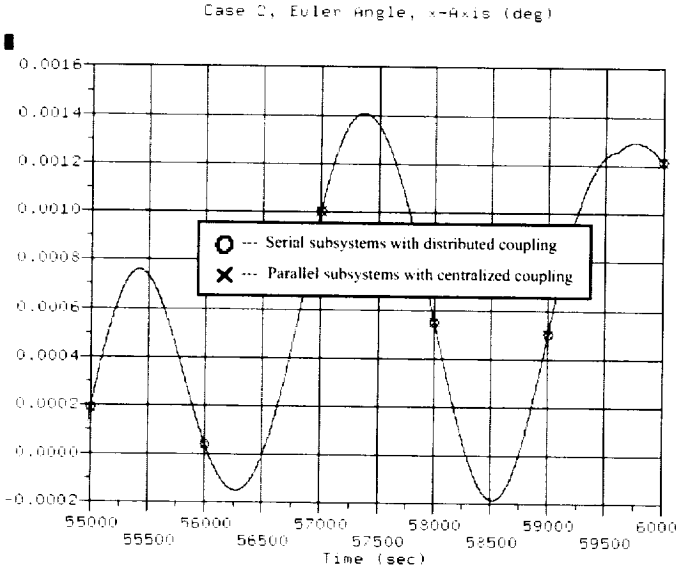

Case 1, Body Rate, y-Axis (deg/sec)
X10^-4
0.15
0.10
0.05
0.00
-0.05
-0.10
-0.15
-0.20
-0.25
59000
59100
59200
59300
59400
59500
59600
59700
59800
59900
60000
Time (sec)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling

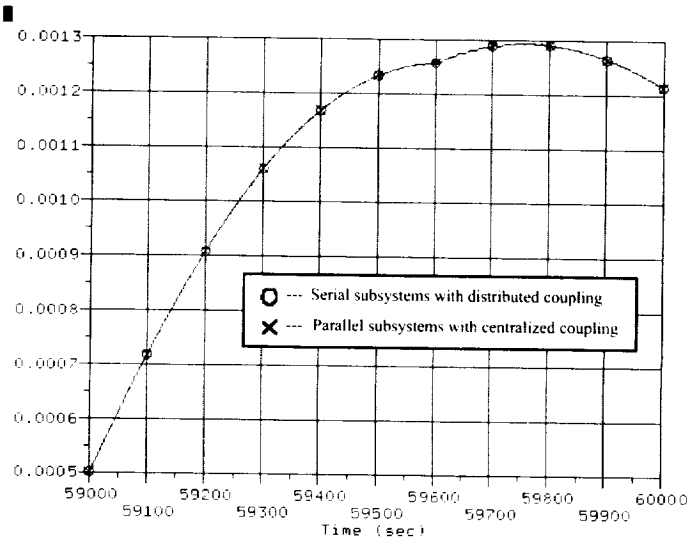

Case 1, Body Rate, z-Axis (deg/sec)
x10^-4
0.6
0.4
0.2
0.0
-0.2
-0.4
-0.6
O --- Serial subsystems with distributed coupling
X --- Parallel subsystems with centralized coupling
50000
51000
52000
53000
54000
55000
56000
57000
58000
59000
60000
Time (sec)

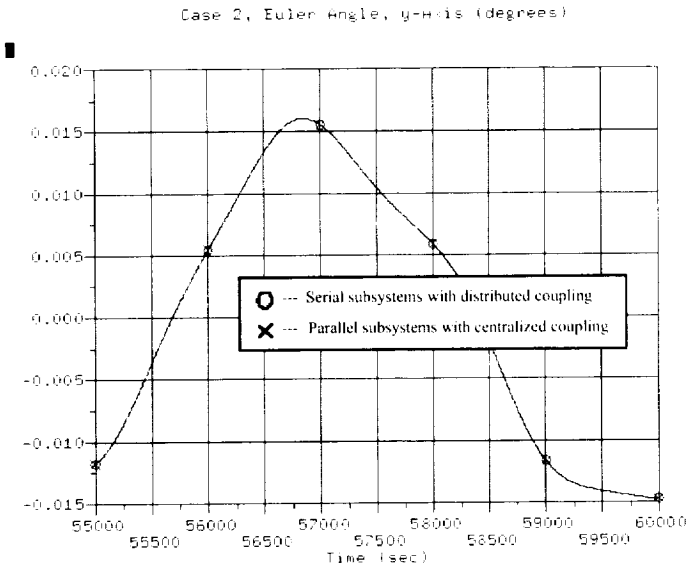

Case 1, Body Rate, z-Axis (deg/sec)
x10^-4
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
O --- Serial subsystems with distributed coupling
X --- Parallel subsystems with centralized coupling
59000
59100
59200
59300
59400
59500
59600
59700
59800
59900
60000
Time (sec)

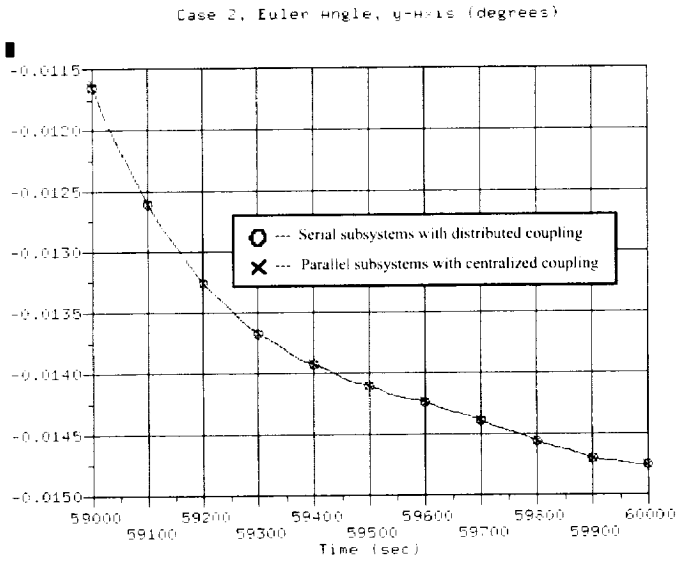

Case 2, Altitude (meters)
299.26
299.25
299.24
299.23
299.22
299.21
299.20
299.19
299.18
O --- Serial subsystems with distributed coupling
X --- Parallel subsystems with centralized coupling
55000
55500
56000
56500
57000
57500
58000
58500
59000
59500
60000
Time (sec)

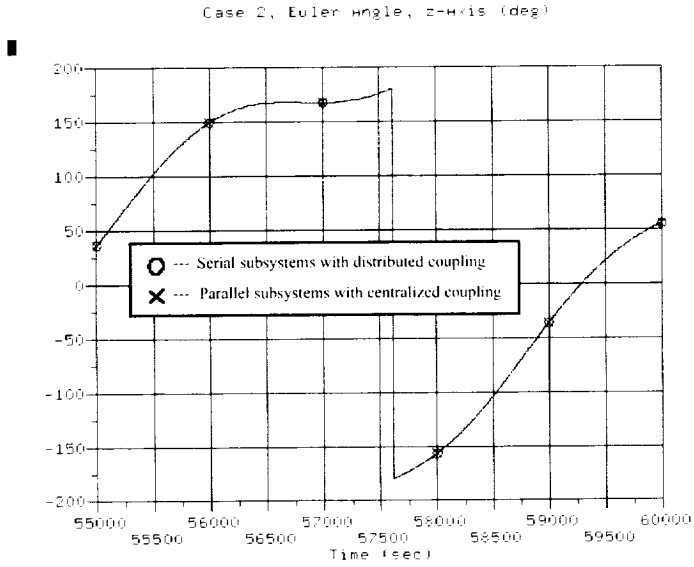

Case 2, Altitude (meters)
299.1830
299.1828
299.1826
299.1824
299.1822
O --- Serial subsystems with distributed coupling
X --- Parallel subsystems with centralized coupling
59000
59100
59200
59300
59400
59500
59600
59700
59800
59900
60000
Time (sec)

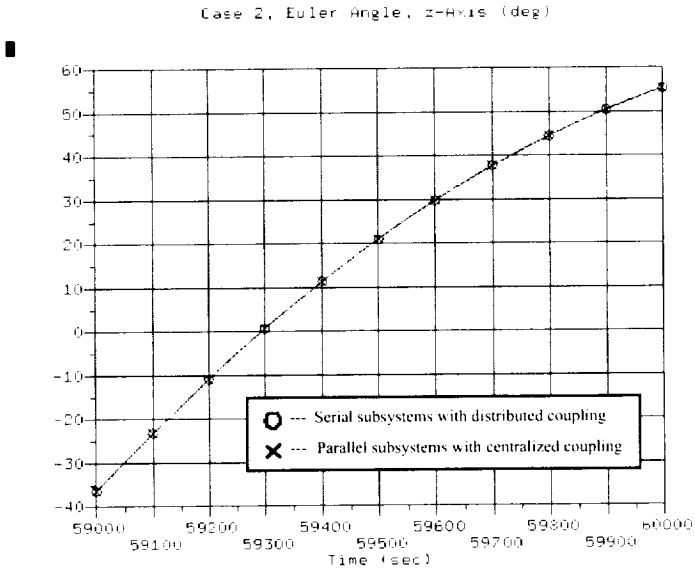

Case 2, Euler Angle, x-Axis (deg)
0.0016
0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
-0.0002
O --- Serial subsystems with distributed coupling
X --- Parallel subsystems with centralized coupling
55000
55500
56000
56500
57000
57500
58000
58500
59000
59500
60000
Time (sec)

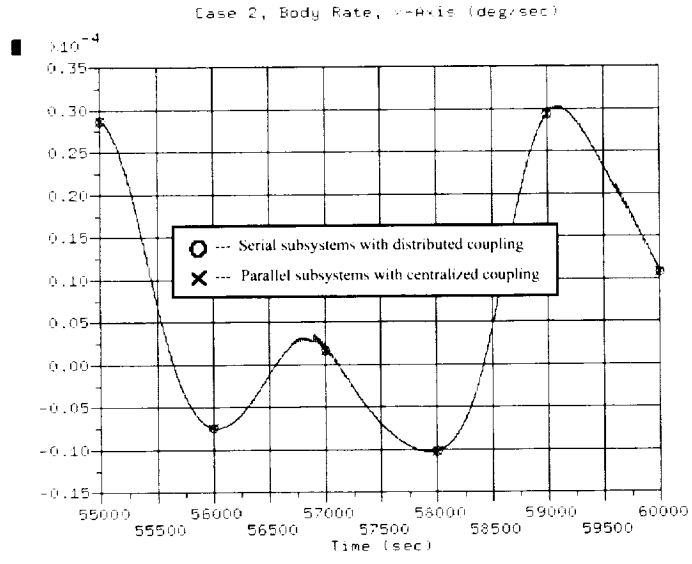

Case 2, Euler Angle, x-Axis (deg)
0.0013
0.0012
0.0011
0.0010
0.0009
0.0008
0.0007
0.0006
0.0005
O --- Serial subsystems with distributed coupling
X --- Parallel subsystems with centralized coupling
59000
59100
59200
59300
59400
59500
59600
59700
59800
59900
60000
Time (sec)

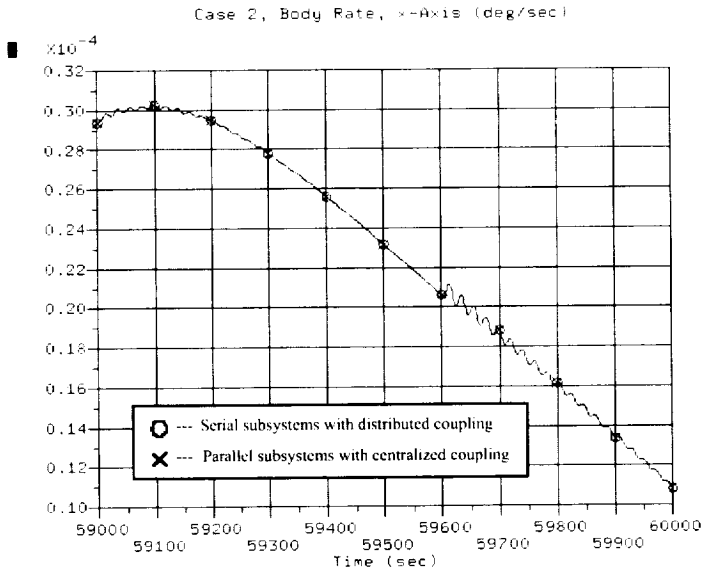

Case 2, Euler Angle, y-Axis (degrees)
0.020
0.015
0.010
0.005
0.000
-0.005
-0.010
-0.015
55000
55500
56000
56500
57000
57500
58000
58500
59000
59500
60000
Time (sec)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling

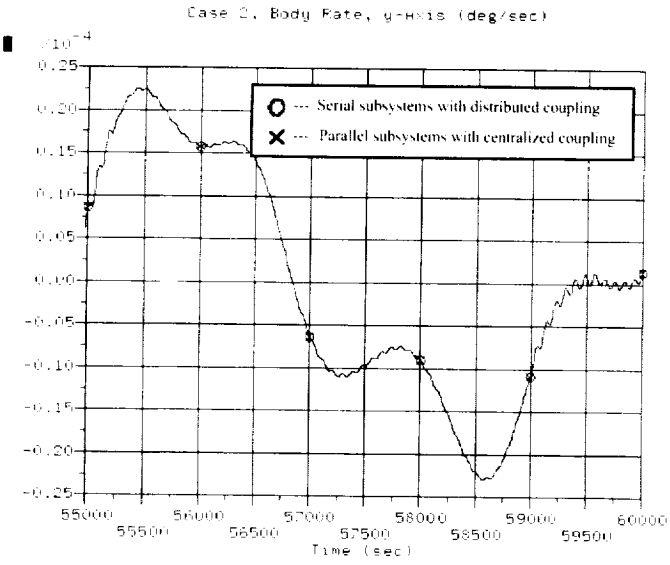

Case 2, Euler Angle, y-Axis (degrees)
-0.0115
-0.0120
-0.0125
-0.0130
-0.0135
-0.0140
-0.0145
-0.0150
59000
59100
59200
59300
59400
59500
59600
59700
59800
59900
60000
Time (sec)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling

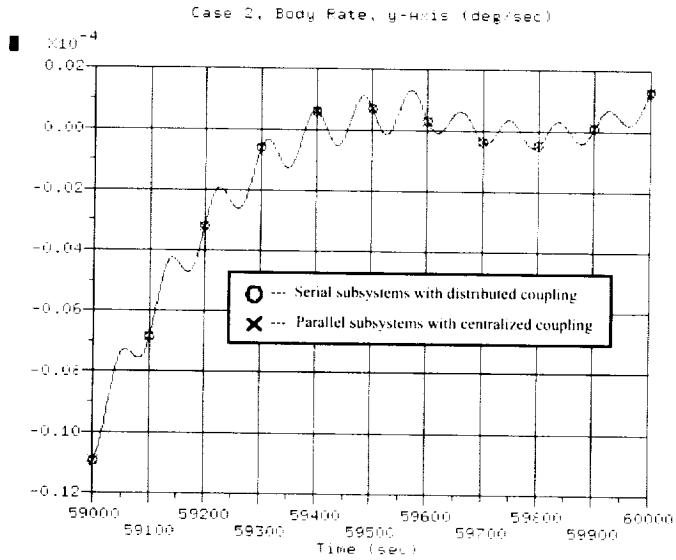

Case 2, Euler Angle, z-Axis (deg)
200
150
100
50
0
-50
-100
-150
-200
55000
55500
56000
56500
57000
57500
58000
58500
59000
59500
60000
Time (sec)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling

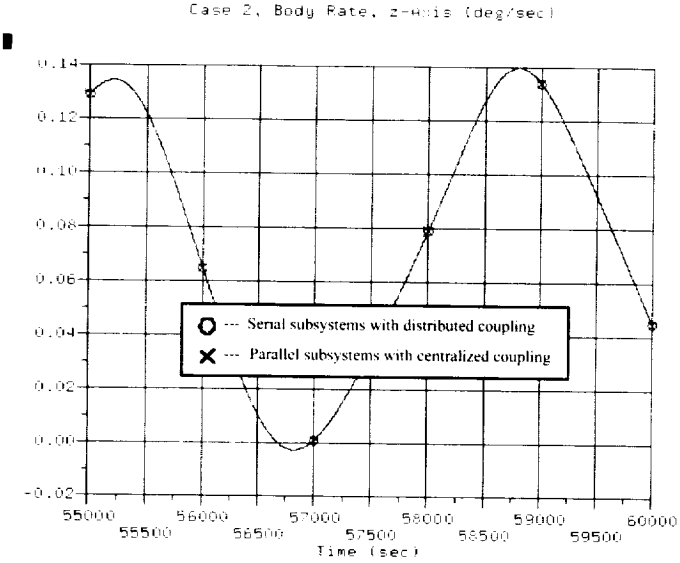

Case 2, Euler Angle, z-Axis (deg)
60
50
40
30
20
10
0
-10
-20
-30
-40
59000
59100
59200
59300
59400
59500
59600
59700
59800
59900
60000
Time (sec)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling

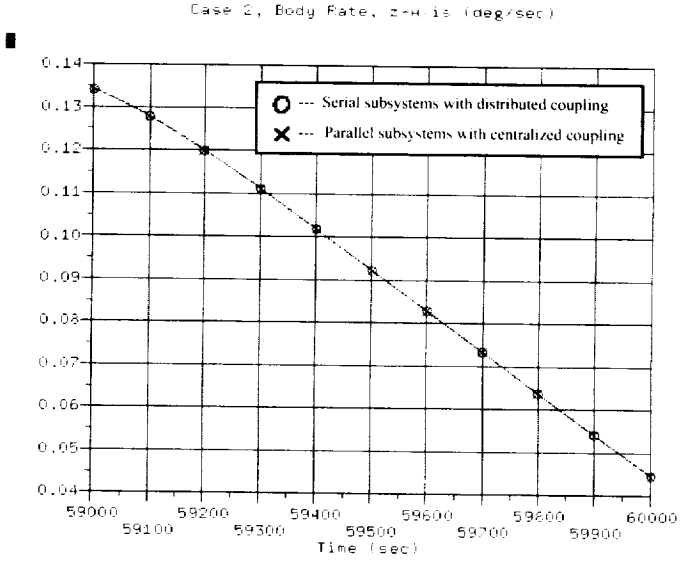

Case 2, Body Rate, x-Axis (deg/sec)
x10^-4
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
-0.05
-0.10
-0.15
55000
55500
56000
56500
57000
57500
58000
58500
59000
59500
60000
Time (sec)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling

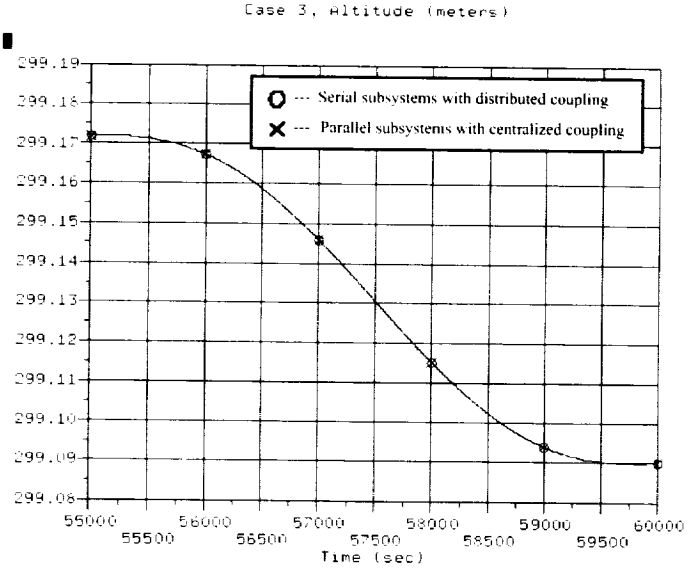

Case 2, Body Rate, x-Axis (deg/sec)
x10^-4
0.32
0.30
0.28
0.26
0.24
0.22
0.20
0.18
0.16
0.14
0.12
0.10
59000
59100
59200
59300
59400
59500
59600
59700
59800
59900
60000
Time (sec)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling

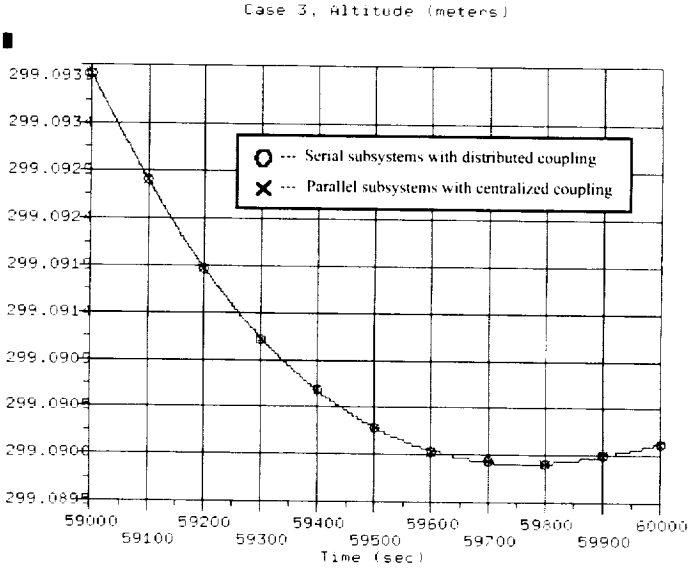

Case 2, Body Rate, y-Axis (deg/sec)
X10^-4
0.25
0.20
0.15
0.10
0.05
0.00
-0.05
-0.10
-0.15
-0.20
-0.25
O --- Serial subsystems with distributed coupling
X --- Parallel subsystems with centralized coupling
55000
55500
56000
56500
57000
57500
58000
58500
59000
59500
60000
Time (sec)

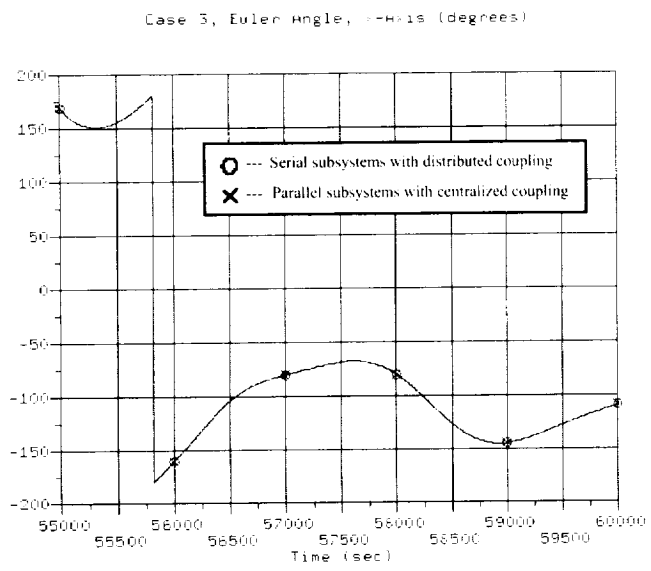

Case 2, Body Rate, y-Axis (deg/sec)
X10^-4
0.02
0.00
-0.02
-0.04
-0.06
-0.08
-0.10
-0.12
O --- Serial subsystems with distributed coupling
X --- Parallel subsystems with centralized coupling
59000
59100
59200
59300
59400
59500
59600
59700
59800
59900
60000
Time (sec)

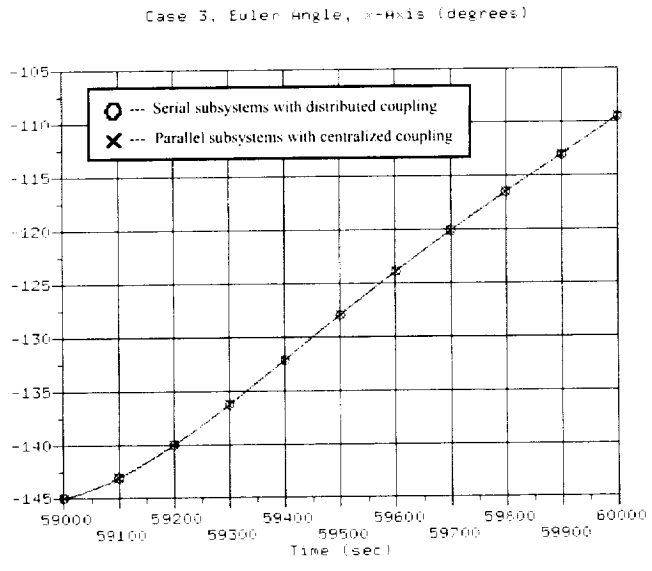

Case 2, Body Rate, z-Axis (deg/sec)
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
-0.02
O --- Serial subsystems with distributed coupling
X --- Parallel subsystems with centralized coupling
55000
55500
56000
56500
57000
57500
58000
58500
59000
59500
60000
Time (sec)

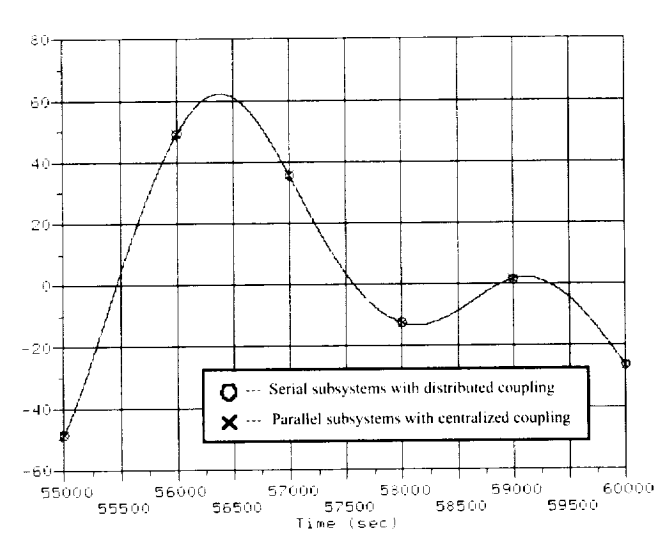

Case 2, Body Rate, z-Axis (deg/sec)
0.14
0.13
0.12
0.11
0.10
0.09
0.08
0.07
0.06
0.05
0.04
O --- Serial subsystems with distributed coupling
X --- Parallel subsystems with centralized coupling
59000
59100
59200
59300
59400
59500
59600
59700
59800
59900
60000
Time (sec)

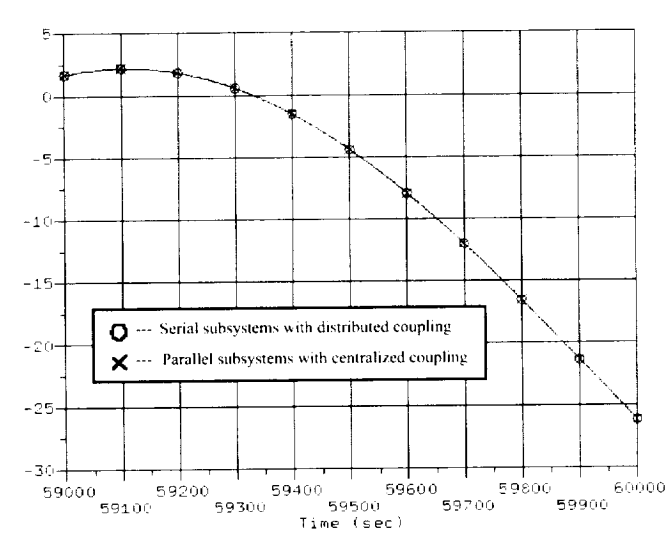

Case 3, Altitude (meters)
299.19
299.18
299.17
299.16
299.15
299.14
299.13
299.12
299.11
299.10
299.09
299.08
O --- Serial subsystems with distributed coupling
X --- Parallel subsystems with centralized coupling
55000
55500
56000
56500
57000
57500
58000
58500
59000
59500
60000
Time (sec)

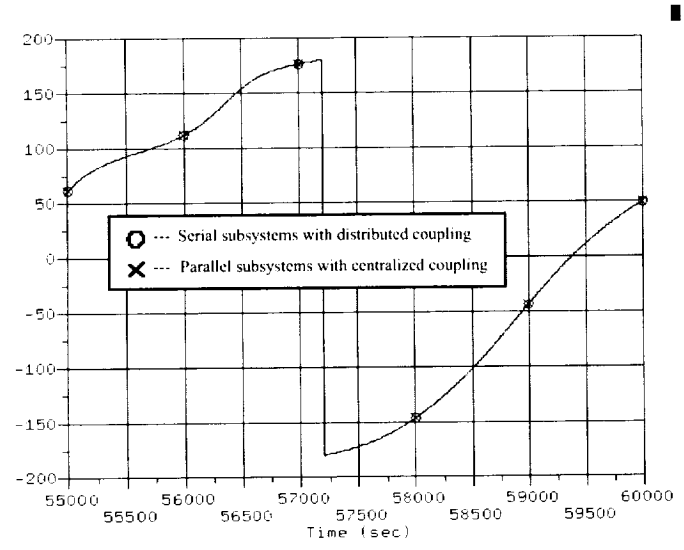

Case 3, Altitude (meters)
299.0933
299.0934
299.0925
299.092
299.0915
299.091
299.0905
299.0900
299.0895
O --- Serial subsystems with distributed coupling
X --- Parallel subsystems with centralized coupling
59000
59100
59200
59300
59400
59500
59600
59700
59800
59900
60000
Time (sec)

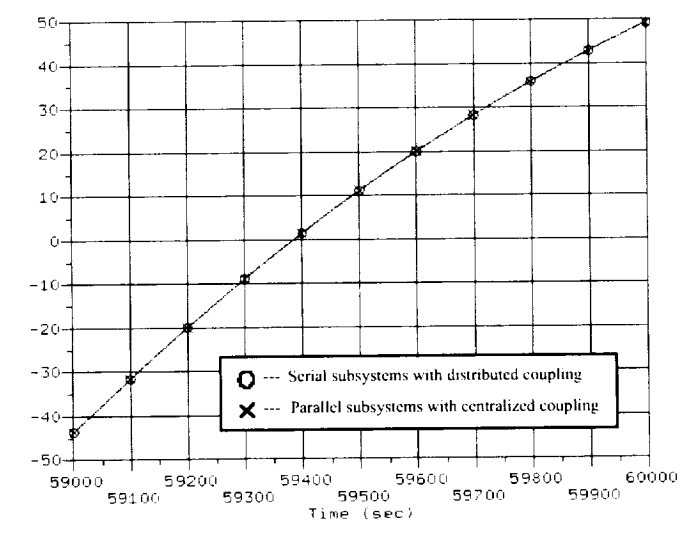

Case 3, Euler Angle, x-Axis (degrees)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling
Time (sec)

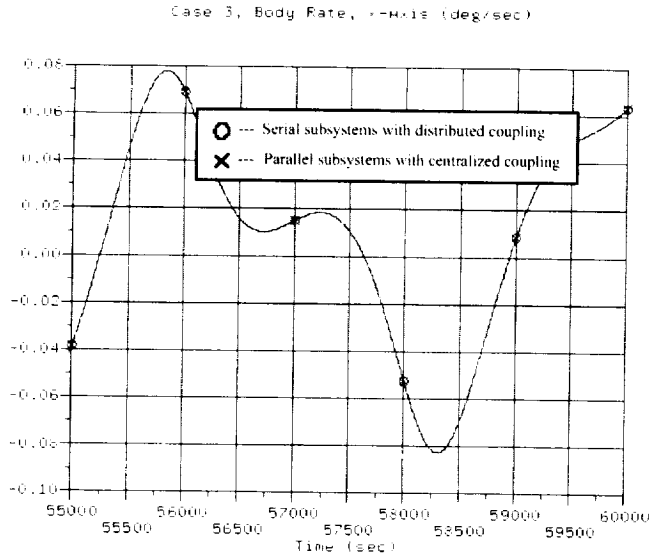

Case 3, Euler Angle, x-Axis (degrees)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling
Time (sec)

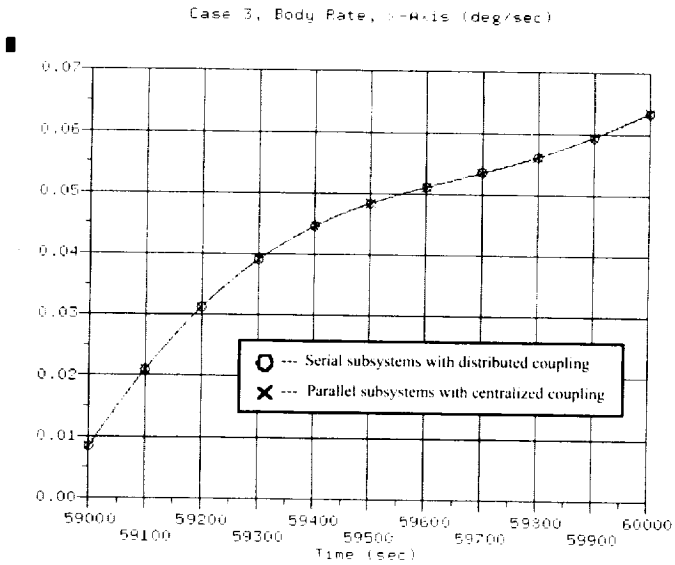

Case 3, Euler Angle, y-Axis (degrees)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling
Time (sec)

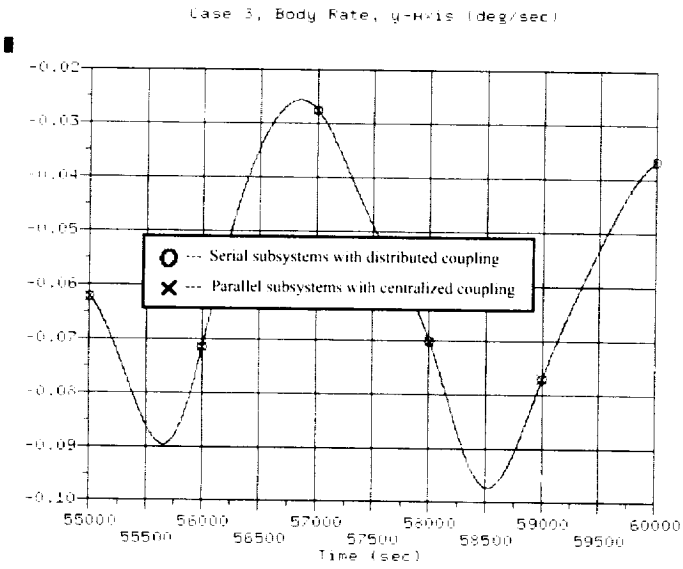

Case 3, Euler Angle, y-Axis (degrees)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling
Time (sec)

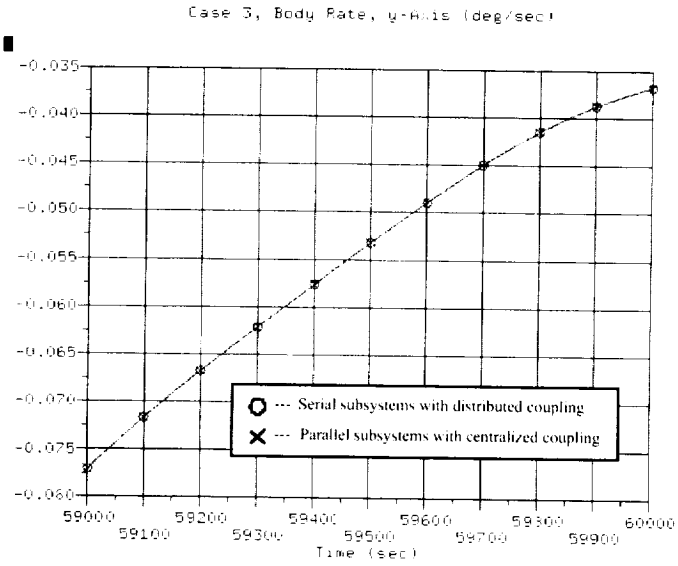

Case 3, Euler Angle, z-Axis (degrees)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling
Time (sec)

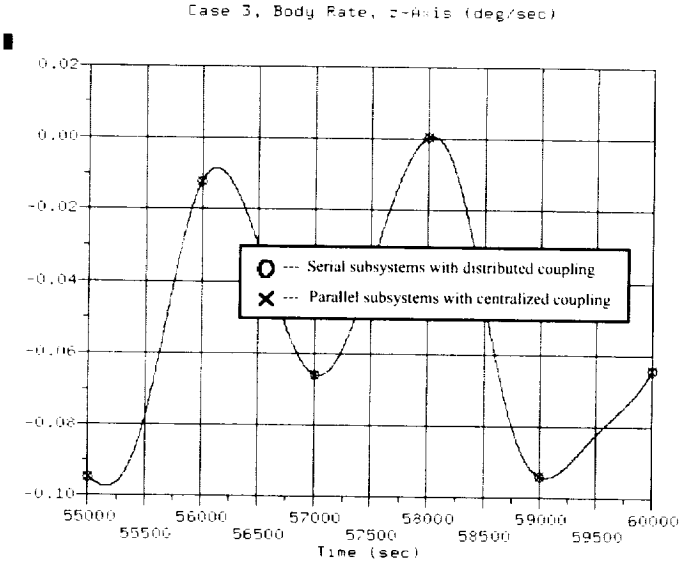

Case 3, Euler Angle, z-Axis (degrees)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling
Time (sec)

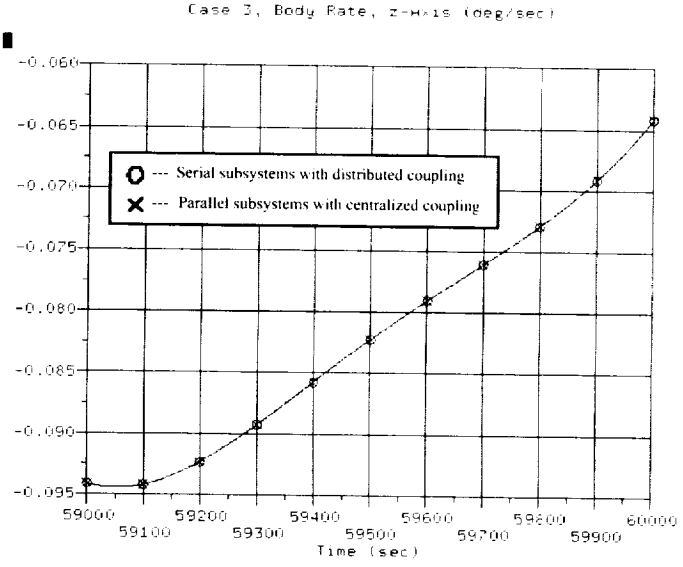

Case 3, Body Rate, x-Axis (deg/sec)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling
Time (sec)


Case 3, Body Rate, x-Axis (deg/sec)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling
Time (sec)


Case 3, Body Rate, y-Axis (deg/sec)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling
Time (sec)


Case 3, Body Rate, y-Axis (deg/sec)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling
Time (sec)


Case 3, Body Rate, z-Axis (deg/sec)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling
Time (sec)


Case 3, Body Rate, z-Axis (deg/sec)
--- Serial subsystems with distributed coupling
--- Parallel subsystems with centralized coupling
Time (sec)

APPENDIX B—FORTRAN LISTINGS

B.1 FORTRAN Listings, Including Main Program and All Subroutines, Recursive Feedback With Centralized Coupling (RFC)

```
c
c     This is FORTRAN main program "rfc.f". Development began 6/12/2000 by Doug Nixon, MSFC/TD55,
c     in support of AEI project, and the listing is contained in "aei/rfc"
c
c     The purpose is to generate the dynamic solution of three coupled subsystems (orbit dynamics,
c     attitude dynamics, and aerodynamics) by recursive feedback, which involves distributed and
c     possibly multiple methods of numerical integration. The intent is to compare results generated
c     by a standard approach using a centralized single numerical integration method. The standard
c     approach has been implemented as FORTRAN program "rku.f", and the main program is contained
c     in "aei/rku"
c
c *** Declarations ************************************************************************
c
      implicit double precision(a-h,o-z)
c
c
      dimension uu(18,100),yy(18,100),yyp(18,100)     !18 input and 18 output signals, 100 time points
c
      dimension uu1(6,100),yy1(6,100)                 !input and output signals for three subsystems,
      dimension uu2(6,100),yy2(6,100)                 !six signals per subsystem, 100 time points
      dimension uu3(6,100),yy3(6,100)
c
      dimension yy1i(6),yy2i(6),yy3i(6)               !initial conditions, 3 subsystems, 6 outputs
      dimension yy1f(6),yy2f(6),yy3f(6)               !final conditions, 3 subsystems, 6 outputs
c
      common/system/uu,yy,yyp      !system input/output/"previous" output, all vs. time
c
      common/subsys1/uu1,yy1       !orbit dynamics, i/o vs. time
      common/endcond1/yy1i,yy1f    !initial and final output conditions
c
      common/subsys2/uu2,yy2       !attitude dynamics, i/o vs. time
      common/endcond2/yy2i,yy2f    !initial and final output conditions
c
      common/subsys3/uu3,yy3       !aerodynamics, i/o vs. time
      common/endcond3/yy3i,yy3f    !initial and final output conditions
c
      common/simulation/ndp,delt,tol,res,t_start,     !all except some single-time-point utility routines
     1t_report,t_print,t_stop,nmt,nmtr,nit,
```

```
     2t_initial,t_final
c
      common/simulation2/del_report,del_print,ptol     !percent done report interval, print interval
c
      common/constants/rearth,gme,pi,cvdr              !recursive.f, orbit.f,sub1, attitude.f, sub2.f
c
c *** Definition of the fundamental variables ************************************************************
c
c     uu(i,j) ----- System-level feedback signal. Represents the ith component at the jth time
c                   point for the total system vector feedback signal. Dimension is that of the
c                   total system, and all subsystem input vectors are contained within appropriate
c                   partitions. Data for the first subsystem is contained in the first (top)
c                   sub-column. In a more general setting, this signal would be summed with a
c                   system-level input signal vv(i,j) to form the subsystem input signals. vv(i,j)
c                   is not needed for this implementation, and has not been included.
c
c     yy(i,j) ----- System-level output signal. Represents the ith component at the jth time
c                   point for the system vector output signal. Dimension is that of the total
c                   system, and all subsystem output vectors are contained within appropriate
c                   partitions. Data for the first subsystem is contained in the first (top)
c                   partition.
c
c ********************************************************************************************************
c
c     uu1(i,j)----- First subsystem input signal. Represents the ith component at the jth time
c                   point for the vector input signal to the first (orbit dynamics) susystem,
c                   and is also the first (top) partition of uu(i,j).
c
c     uu2(i,j)----- Second subsystem input signal. Represents the ith component at the jth time
c                   point for the vector input signal to the second (attitude dynamics) subsystem,
c                   and is also the second (middle) partition of uu(i,j).
c
c     uu3(i,j)----- Third subsystem input signal. Represents the ith component at the jth time
c                   point for the vector input signal to the third (aerodynamics) subsystem, and
c                   is also the third (bottom) partition of uu(i,j).
c
c ********************************************************************************************************
cc     yy1(i,j)----- First subsystem output signal. Represents the ith component at the jth time
```

```
c                  point for the vector output signal from the first (orbit dynamics) subsystem,
c                  and is also the first (top) partition of yy(i,j).
c
c     yy2(i,j)----- Second subsystem output signal. Represents the ith component at the jth time
c                  point for the vector output signal from the second (attitude dynamics) subsystem,
c                  and is also the second (middle) partition of yy(i,j).
c
c     yy3(i,j)----- Third subsystem output signal. Represents the ith component at the jth time
c                  point for the vector output signal to the third (aerodynamics) subsystem, and is
c                  also the third (bottom)partition of yy(i,j).
c
c ***************************************************************************************************
c
c     yy1i(i) ----- Value of the output signal at the beginning of the current convergence interval
c                  for the FIRST (orbit dynamics) subsystem. Represents the ith component of the
c                  signal at the first time point, and is equivalent to yy1(i,1)
c
c     yy2i(i) ----- Value of the output signal at the beginning of the current convergence interval
c                  for the SECOND (attitude dynamics) subsystem. Represents the ith component of the
c                  signal at the first time point, and is equivalent to yy2(i,1)
c
c     yy3i(i) ----- Value of the output signal at the beginning of the current convergence interval
c                  for the THIRD (aerodynamics) subsystem. Represents the ith component of the signal
c                  at the first time point, and is equivalent to yy3(i,1)
c ***************************************************************************************************
c
c     yy1f(i) ----- Value of the output signal at the end of the current convergence interval for
c                  the FIRST (orbit dynamics) subsystem. Represents the ith component of the signal
c                  at the last time point, and is equivalent to yy1(i,ndp)
c
c     yy2f(i) ----- Value of the output signal at the end of the current convergence interval for
c                  the SECOND (attitude dynamics) subsystem. Represents the ith component of the
c                  signal at the last time point, and is equivalent to yy2(i,ndp)
c
c     yy3f(i) ----- Value of the output signal at the end of the current convergence interval for
c                  the THIRD (aerodynamics) subsystem. Represents the ith component of the vector
c                  signal yy3(i,j) at the last time point, and is equivalent to yy3(i,ndp)
c
```

```
c ******General Comments ************************************************************************
c
c     The above nomenclature has been designed to allow implementation of recursive feedback via
c     FORTRAN in such a way that independence of the subsystem simulations can be supported. The
c     partitions and contractions facilitate autonomy of the subsystem simulations while providing
c     appropriate structure for implementation of the system-level algorithm. The fact that subsystem
c     data has been placed in COMMON with the MAIN (system level) program does not necessarily mean
c     that all of that data is required or desired at the system level -- it may mean only that the
c     SUBROUTINE containing the subsystem simulation needs to have access to data that it generated
c     previously, and placing that data in COMMON is a way to provide that in FORTRAN.
c
c     While the subsystem simulation SUBROUTINES will not literally be executed in parallel, the
c     information flows in parallel paths, and the order of execution is not important. Therefore,
c     parallel execution is conceptually possible without impact to the structure of the process.
c
c     This program, which involves three subsystems, can serve as a template for implementation with any
c     number of subsystems. Use of higher order arrays to implement the general case for "n" subsystems
c     appears to serve no useful purpose in this setting at this time.
c
c     This implementation assumes that all subsystem states are needed at the system level, and transfer of
c     final conditions to initial conditions after convergence is handled at that level. For this specific
c     system, the assumption is valid; however, in general,subsystem states can exist and remain at the
c     subsystem level without the necessity of knowing them at the system level. Because of this,
c     it would be more appropriate to define and "bookkeep" initial conditions at the subsystem level for a
c     more general implementation. In the more general implementation, transfer of final conditions from
c     one convergence interval to intial conditions of the next would be handled totally at the subsystem
c     level upon receipt of "command" from the system level.
c
c *** Establishing the output file ***************************************************************
c
      open(unit=1,file='rfc.out1',status='unknown')
c
c *** Defining system-level simulation parameters ************************************************
c
      print*
      print*,'This is main program rfc.f. Solution'
      print*,'method is recursive feedback with'
      print*,'centralized coupling.'
```

```
      print*
      print*,'  Input the number of data (time) points'
      print*,'  per convergence interval, including'
      print*,'  both end (boundary) points:'
      read*, ndp
      print*,'  Input the convergence interval duration'
      print*,'  (sec.):'
      read*, dtmaj
c
      fac=1.0/float(ndp-1)      !ratio of integration time step to convergence interval duration
      delt=fac*dtmaj            !integration stepsize
c
      print*,'  The above data limits all integration'
      print*,'  stepsizes to ',delt,' sec.'
      print*,'  or less'
      print*
c
c *** Populating the subsystem databases ***************************************************************
c
      call orbit(1)      !defines parameter values (data) for subsystem 1 (orbit dynamics, mode 1)
c
      call attitude(1)   !defines parameter values (data) for subsystem 2 (orbit dynamics, mode 1)
c
      call aerodyn(1)    !defines parameter values (data) for subsystem 3 (orbit dynamics, mode 1)
c
      print*,'  Input the simulation stop time (sec.):'
      read*, t_stop
      print*,'  Input the data output(print) interval'
      print*,'  (sec.):'
      read*, del_print
      print*,'  Input percent done status interval'
      print*,'  (%):'
      read*, per_report
      del_report=(per_report/100.0)*t_stop
c
c
      nmtr=int(t_stop/dtmaj)+1   !number of major time (convergence) intervals required for completion of
c                                !the simulation. (Time may exceed t_stop by as much as one major interval)
```

```
c
      tol=(1.0D0)*1.0e-12        !relative
      res=(1.0D0)*1.0e-12        !absolute
      itmax=100                  !maximum allowable number of iterations
      ptol=1.0e-3                !tolerance for print time
      print*,'  Run in progress.'
      print*
      call clock(n1)            !machine start time
c
c *** Initializing time and convergence interval count: ************************************************
c
      time=0.0    !time is measured from start of the simulation
      nmt=0       !number of major time intervals (convergence intervals) simulated
      t_print=0.0  !initializes print time
      t_report=0.0 !initializes percent done report time
c
c *** Setting up initial conditions for each sub-simulation at system simulation outset
c
      call orbit(2)     !mode 2, original initial conditions setup, orbit dynamics
c
      call attitude(2)  !mode 2, original initial conditions setup, attitude dynamics
c
c                       !no initial conditions setup for aerodynamics (algebraic system)
c
c ***************************************************************************************************
  100 continue !*****************COMPUTING INITIAL RECURSION STAGE DATA*********************************
c ***************************************************************************************************
c
      nit=0 !number of iterations initialized to zero for first (new) convergence interval
c
c *** Initializing the system-level feedback signal to zero: ******************************************
c
      do 1 i=1,18    !eighteen scaler feedback signals
      do 1 j=1,ndp   !ndp number of data (time) points per scaler signal
      uu(i,j)=0.0D0  !algorithm ignores feedback at recursion stage number (0)
    1 continue
c
c *** Decomposing the system-level feedback vector into individual subsystem input vectors
```

```
c
      call decomp(6,ndp,uu,uu1,uu2,uu3) !six scaler input signals per subsystem
c
c *** Running individual subsystem simulations over the convergence interval
c
      call orbit(3)     ! mode 3, run orbit dynamics simulation over convergence interval
c
      call attitude(3)  ! mode 3, run attitude dynamics simulation over convergence interval
c
      call aerodyn(3)   ! mode 3, run aerodynamics (algebraic) simulation over convergence interval
c
c *** Assemble subsystem outputs into single system-level output ***************************************
c
      call assemble(6,ndp,yy1,yy2,yy3,yy)
c
      call coupler(yy,uu)  !computes system-level feedback signal (subsystem coupling)

      do 2 i=1,18       !sets initial recursion stage data to current values
      do 2 j=1,ndp
      yyp(i,j)=yy(i,j)
    2 continue
c
c ********************************************************************************************************
  200 continue !*********************RECURSION LOOP ENTRY POINT*****************************************
c ********************************************************************************************************
c
c *** Decomposing the system-level feedback vector into individual subsystem input vectors
c
      call decomp(6,ndp,uu,uu1,uu2,uu3) !six scaler input signals per subsystem
c
c *** Running individual subsystem simulations over the convergence interval
c
      call orbit(3)     ! mode 3, run orbit dynamics simulation over convergence interval
c
      call attitude(3)  ! mode 3, run attitude dynamics simulation over convergence interval
c
      call aerodyn(3)   ! mode 3, run aerodynamics (algebraic) simulation over convergence interval
```

```
c
c *** Assemble subsystem outputs into single system-level output ***************************************
c
      call assemble(6,ndp,yy1,yy2,yy3,yy)
c
c *** Check for convergence ***************************************************************************
c
      call converge(18,yy,yyp,iconflag)       ! system-level convergence check
c                                              ! iconflag=0 if converged, 1 if
c                                              ! notconverged
      idirect=iconflag+1
  500 format(/,2i5,/)
      go to (6,5) idirect          !6 for converged, 5 for not converged
c
c ********************************************************************************************************
    5 continue ! ********** Not CONVERGED, ADVANCE RECURSION STAGE ********************************
c ********************************************************************************************************
c
      call coupler(yy,uu)  !computes system-level feedback signal (subsystem coupling)

      do 7 i=1,18          !number of scaler signals in the system
      do 7 j=1,ndp         !number of time points over each convergence interval,
c                          !includes both end points
      yyp(i,j)=yy(i,j)     !sets previous recursion stage data to current values
    7 continue
      nit=nit+1            !updating the iterations count (number of recursion stages)
c
      if(nit.ge.itmax) go to 300     !protection against persistent non convergence
c
      go to 200            !returns to "top" of the recursion loop to run subsystem simulations again
c
c ********************************************************************************************************
    6 continue ! ** CONVERGED, ADVANCE TO NEXT CONVERGENCE INTERVAL (or stop if simulation is complete) **
c ********************************************************************************************************
c
      call rfc_out           !appends system-level output signals to output file (recursive.out)
      nmt=nmt+1              !increments number of major time (convergence) intervals completed
      time=nmt*dtmaj         !simulated problem time , for reference only
```

```
c
      if(nmt.eq.nmtr) go to 300   !end of run, time is greater than or equal to the stop time (t_stop)
c
c ***************************************************************************************************
c
c *** Proceed ("hand-off") to next major time (convergence) interval
c
c     In a more general setting, this would be done at the subsystem level (mode 4) so that the system
c     level is not required to have knowledge of all subsystem states.
c
      do 8 i=1,6              !extracts subsystem final conditions for the current convergence interval
      yy1f(i)=yy(i,ndp)       !from the final time point of the system response associated with the current
      yy2f(i)=yy(i+6,ndp)     !convergence interval. SEE PARAGRAPH NUMBER 4 of "General Comments" ABOVE.
      yy3f(i)=yy(i+12,ndp)
    8 continue
c
      do 9 i=1,6
      yy1i(i)=yy1f(i)         !sets initial conditions (for all states) for the next convergence interval
      yy2i(i)=yy2f(i)         !equal to conditions reached at the end (final conditions) of current
      yy3i(i)=yy3f(i)         !convergence interval.
    9 continue
c
      go to 100               !run subsystem simulations for the first time across the new (next)
c                             !convergence interval.
  300 continue
c
      if(nit.ge.itmax) print*,'Max. allowable number of '
      if(nit.ge.itmax) print*,'iterations reached.'
c
      call clock(n2) !machine stop time
      irun=n2-n1     !run-time

      if(nmt.eq.nmtr) print*,'  Normal exit, run-time = ',irun,' sec.'
      print*
c
      close(unit=1,status='keep')
c
      stop
```

```
c
c *********************************************************************************************************
      end !********************************** END OF MAIN PROGRAM *****************************************
c *********************************************************************************************************

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx


      subroutine aerocompr(vel_b,force,torque)
c
c     Developed by Doug Nixon, TD55, MSFC starting 5/26/2000 to compute
c     aerodynamic forces and torques for the "truth model" for recursive
c     feedback simulation method comparison for AEI study.
c
c     Modified by Doug Nixon, TD55, MSFC from standard/aerocomp.f starting 6/21/2000 for operation
c     in the recursive feedback setting. The attitude DCM was removed from the argument because the
c     inertial to body transformation is now performen by the system-level coupler.


c
c     rho        = Atmospheric density, kg/m**3
c     vel_b(i)   = Velocity vector in body coordinates, meters/sec
c     force(i)   = Aerodynamic force, body coordinates, newtons
c     torque(i)  = Aerodynamic torque, body coordinates, newton-meters
c
      implicit double precision (a-h,o-z)
c
      dimension vel_b(3),force(3),torque(3)
      dimension vnorm_b(3)
      dimension cpfront(3),cpright(3),cpbottom(3)
      dimension cprear(3),cpleft(3),cptop(3)
      dimension afront(3),aright(3),abottom(3)
      dimension arear(3),aleft(3),atop(3)
```

```
      dimension aproj(6),drag(6),facefor(6,3),facetor(6,3)
      dimension tforce(3),ttorq(3)
c
      common/aerobox/cpfront,cpright,cpbottom,        !aerodyn.f, aerocompr.f
     1cprear,cpleft,cptop,afront,aright,abottom,
     2arear,aleft,atop
c
      common/aerodat/rho,cd                           !aerodyn.f, aerocompr.f
c
c*************************************************************************************************

c
c *** Aerodynamic forces ***
c
      call vecnorm(vel_b,vnorm_b,vmag)      ! normalize velocity vector in body
c                                           ! coordinates
c
c *** computing magnitude drag for each of six facets
c
      call vdotpro(afront,vnorm_b,aproj(1))    ! computing dot product of each area
      call vdotpro(aright,vnorm_b,aproj(2))    ! vector with normalized velocity
      call vdotpro(abottom,vnorm_b,aproj(3))   ! (projected areas)
      call vdotpro(arear,vnorm_b,aproj(4))
      call vdotpro(aleft,vnorm_b,aproj(5))     ! these statements define the facet
      call vdotpro(atop,vnorm_b,aproj(6))      ! numbering system
c
      do 20 i=1,6
      if(aproj(i).le.0.0D0) aproj(i)=0.0D0   ! negative projection implies ith facet
   20 continue                               ! not exposed to velocity vector
c
      fac=(0.5D0)*rho*(vmag**2)*cd           ! scaling by dynamic pressure times
      do 21 i=1,6                            ! drag coefficient
      drag(i)=fac*aproj(i)
   21 continue
c
      do 22 i=1,6
      do 22 j=1,3
      facefor(i,j)=-drag(i)*vnorm_b(j)       ! this is jth component of force on the ith
```

```
   22 continue                                    ! facet, expressed in body coordinates
c
      do 23 i=1,3
      force(i)=0.0D0
      do 23 j=1,6
      force(i)=force(i)+facefor(j,i)  ! total aerodynamic force, body coordinates, newtons
   23 continue
c
c *** Aerodynamic Torques ********************************************************************
c
      do 30 i=1,3                              ! front facet contribution
      tforce(i)=facefor(1,i)
   30 continue
      call vcross(cpfront,tforce,ttorq)
      do 31 i=1,3
      facetor(1,i)=ttorq(i)
      torque(i)=torque(i)+ttorq(i)
   31 continue
c
      do 32 i=1,3                              ! adding right facet contribution
      tforce(i)=facefor(2,i)
   32 continue
      call vcross(cpright,tforce,ttorq)
      do 33 i=1,3
      facetor(2,i)=ttorq(i)
   33 continue
c
      do 34 i=1,3                              ! adding bottom facet contribution
      tforce(i)=facefor(3,i)
   34 continue
      call vcross(cpbottom,tforce,ttorq)
      do 35 i=1,3
      facetor(3,i)=ttorq(i)
   35 continue
c
      do 36 i=1,3                              ! adding rear facet contribution
      tforce(i)=facefor(4,i)
   36 continue
```

```
      call vcross(cprear,tforce,ttorq)
      do 37 i=1,3
      facetor(4,i)=ttorq(i)
   37 continue
c
      do 38 i=1,3                          ! adding left facet contribution
      tforce(i)=facefor(5,i)
   38 continue
      call vcross(cpleft,tforce,ttorq)
      do 39 i=1,3
      facetor(5,i)=ttorq(i)
   39 continue
c
      do 40 i=1,3                          ! adding top facet contribution
      tforce(i)=facefor(6,i)
   40 continue
      call vcross(cptop,tforce,ttorq)
      do 41 i=1,3
      facetor(6,i)=ttorq(i)
   41 continue
c
      do 42 i=1,3
      torque(i)=0.0D0
      do 42 j=1,6
      torque(i)=torque(i)+facetor(j,i)  ! total aerodynamic torque, body
   42 continue                          ! coordinates, newton-meters
c
      return
      end
```

xxxx

```
      subroutine aerodyn(mode)    ! mode 1= set physical constants, system  parameters,etc.
                                  ! mode 2= blank, no initial conditions for algebraic system
                                  ! mode 3= run simulation over the convergence interval
c
c *** This is subprogram "aerodyn.f". Construction began 6/21/2000 by Doug Nixon, TD55, MSFC.
c     Its purpose is to generate aerodynamic forces and torques as a function of time when
c     given an orbit state vector and vehicle attitude definition as a function of time. It is
c     an algebraic system; no numerical integration is involved.

      implicit double precision (a-h,o-z)
c
      dimension uu3(6,100),yy3(6,100)
c
      dimension rad_b(3),vel_b(3)
      dimension force(3),torque(3)
c
      dimension cpfront(3),cpright(3),cpbottom(3)      !box panel data
      dimension cprear(3),cpleft(3),cptop(3)
      dimension afront(3),aright(3),abottom(3)
      dimension arear(3),aleft(3),atop(3)
c
      common/subsys3/uu3,yy3                           !recursive.f, attitude.f, sub2.f
c
      common/endcond3/yy3i,yy3f                        !recursive.f, attitude.f
c
      common/simulation/ndp,delt,tol,res,t_start,      !all except some single-time-point utility routines
     1t_report,t_print,t_stop,nmt,nmtr,nit,
     2t_initial,t_final
c
      common/aerobox/cpfront,cpright,cpbottom,         !aerodyn.f, aerocompr.f
     1cprear,cpleft,cptop,afront,aright,abottom,
     2arear,aleft,atop
c
      common/aerodat/rho,cd                            !aerodyn.f, aerocompr.f
c
      if(mode.eq.2.or.mode.eq.3) go to 200
```

```
c
  100 continue ! (mode=1, define constants, system parameters)
c
c ********************* Setting Values of System Parameters (mode 1 function): ************************
c
c *** Density and drag coefficient: ***
c
      rho=1.95E-11 ! kg/m**3, Wertz,inside back cover, 300km altitude , mean value
c
      cd=2.25                ! drag coefficient, same for all facets
c
c *** "Panel" aerodynamics setup (six-panel box): ***
c
      xl=15.0D0              ! length, meters (box shape, x-axis dimension)
      w=4.0D0                ! width, y-axis dimension
      h=3.0D0                ! height, z-axis dimension
      ax=w*h                 ! end area, normal to x-axis (+ "front")
      ay=xl*h                ! side area, normal to y-axis (+ "right side")
      az=xl*w                ! top or bottom area, normal to (+ z-axis "bottom")
c
c     Work with six facets ----
c
      do 10 i=1,3       ! setting all vector components to 0.0D0
c
      afront(i)=0.0D0   ! area vectors, normal to facet surfaces, through
      arear(i)=0.0D0    ! centers, positive (point) outward from center of
      aright(i)=0.0D0   ! box, back sides not considered (no contribution)
      aleft(i)=0.0D0
      atop(i)=0.0D0
      abottom(i)=0.0D0
c
      cpfront(i)=0.0D0 ! centers of pressure location vectors, from geometric
      cprear(i)=0.0D0  ! center of box to center of each facet
      cpright(i)=0.0D0
      cpleft(i)=0.0D0
      cptop(i)=0.0D0
      cpbottom(i)=0.0D0
c
```

```
   10 continue
c
      afront(1)=ax   ! setting non-0.0D0 components of facet area vectors
      arear(1)=-ax
      aright(2)=ay
      aleft(2)=-ay
      abottom(3)=az
      atop(3)=-az
c
      cpfront(1)=xl/2.0D0    !setting non-zero components of facet
      cprear(1)=-xl/2.0D0    !centers-of-pressure location vectors
      cpright(2)=w/2.0D0
      cpleft(2)=-w/2.0D0
      cptop(3)=-h/2.0D0
      cpbottom(3)=h/2.0D0
c
      return
c
c ***************************************************************************************************
c ******************************** End of mode 1, beginning of mode 2 ********************************
c ***************************************************************************************************
c
  200 if(mode.eq.3) go to 300
      continue ! (mode=2, normally sets initial conditions for dynamic system)
c
      print*,' '
      print*,'This sub-simulation (aerodynamics) is'
      print*,'algebraic; therefore initial conditions'
      print*,'are not applicable.'
      print*,' '
      print*,'Enter 0 to continue, 1 to stop'
      read*, igo
      if(igo.eq.1) stop
c
      return
c
c ***************************************************************************************************
c ******************************** End of mode 2, beginning of mode 3 ********************************
```

```
c ***************************************************************************************************
c
c
  300 continue ! (mode=3, run simulation over the convergence interval)
c
c ***************************** Propagate Output Signals (mode 3 function): ****************************
c
c
c
c *** begin propagation loop
c
      do 20 ip=1,ndp        ! propagates forces and torques across convergence interval,
c
      do 6 j=1,3
      rad_b(j)=uu3(j,ip)    ! extracts radius and velocity vectors at the ith time point from the
      vel_b(j)=uu3(j+3,ip)  ! subsystem 3 input signal array. Radius is expressed in inertial coordinates,
    6 continue              ! velocity in body coordinates. Radius is not used now because density is
c                           ! defined to be constant --- will be used in later version.
c
      call aerocompr(vel_b,force,torque) ! computes force and torque at a single time point
c

      do 7 j=1,3            ! enters output (force and torque vectors) into block common /susbsy1/.
      yy3(j,ip)=force(j)
      yy3(j+3,ip)=torque(j)
    7 continue
c
   20 continue
c
c *** end propagation loop
c
      return
c
c ***************************************************************************************************
c ************************************* End of mode 3 ***********************************************
c ***************************************************************************************************
c
      end
```

```
      subroutine assemble(nsubdim,ndp,xx1,xx2,xx3,xx)
c
c *** Assembles subsystem input or output signals into system-level input or output signals
c
      implicit double precision (a-h,o-z)
c
      dimension xx1(6,100),xx2(6,100),xx3(6,100),xx(18,100)
c
c     Assumes nsubdim number of scalar signals per subsystem
c
      do 10 i=1,nsubdim              !eighteen scaler signals at the systemlevel,
      do 10 j=1,ndp                  !ndp time points each
      xx(i,j)=xx1(i,j)
      xx(i+6,j)=xx2(i,j)
      xx(i+12,j)=xx3(i,j)
   10 continue
c
      return
      end

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      subroutine attitude(mode) ! mode 1= set physical constants, system parameters,etc.
                                ! mode 2= set initial conditions
                                ! mode 3= run simulation over the convergence interval
c
c *** This is subprogram "attitude.f". Construction began 6/28/2000 by Doug Nixon, TD55, MSFC.
c    "attitude.f" is the main driver for the six-state dynamic system which is integrated
c     by "rksys2.f", a fourth-order Runge-Kutta subroutine. "sub2.f" defines the
c     dynamics in first-order state-variable form.
c
      implicit double precision (a-h,o-z)
```

```
c
      dimension uu2(6,100),yy2(6,100)                !I/O for subsystem 2 (attitude dynamics)
c
      dimension yy2i(6),yy2f(6)                      !initial and final output conditions, subsystem 2
c
      dimension x(6),xdot(6)                         !state variables for attitude dynamics
c
      dimension phi(3),omega(3)                      !attitude angles and body rates (state variables)
c
      dimension xmoi(3,3),xmoi_inv(3,3),rcm(3)       !moments of inertia, about c.m, c.m. location vector
c                                                    !in geometric body coordinates
c
      common/subsys2/uu2,yy2                         !recursive.f, attitude.f, sub2.f
c
      common/endcond2/yy2i,yy2f                      !recursive.f, attitude.f
c
      common/simulation/ndp,delt,tol,res,t_start,    !all except some single-time-point utility routines
     1t_report,t_print,t_stop,nmt,nmtr,nit,
     2t_initial,t_final
c
      common/constants/rearth,gme,pi,cvdr            !recursive.f, orbit.f,sub1, attitude.f, sub2.f
c
      common/mass/xmass,xmoi,xmoi_inv,rcm            !orbit.f, sub1.f, attitude.f, sub2.f, coupler.f
c
      common/gains/a0,a1,b0,b1,c0,c1                 !attitude.f, sub2.f
c
      if(mode.eq.2.or.mode.eq.3) go to 200
c
  100 continue ! (mode=1, define constants, system parameters)
c
c ********************************************************************************************
c *************************** Numerically Defining Constants (mode 1 function): ***************
c ********************************************************************************************
c
c            Gravity constant (gme) is defined by orbit.f, and is obtained through block
c            common /constants/; pi (3.14xx...) and cvdr (deg. to rad. mult. factor) are
c            are defined and obtained similarly.
c
```

```
c ********************* Setting Values of System Parameters (mode 1 function): ************************
c
c
c            Satellite mass (xmass) is defined by orbit.f; obtained here through common/mass/.
c
c *** Moments of inertia and center of mass:
c
      xmoi(1,1)=11250.0D0              ! spacecraft moments of inertia,
      xmoi(1,2)=0.0D0                  ! kilogram-meters**2
      xmoi(1,3)=0.0D0
      xmoi(2,1)=0.0D0
      xmoi(2,2)=193125.0D0             ! 3m diameter x 15m length uniform-
      xmoi(2,3)=0.0D0                  ! density right-circular cylinder
      xmoi(3,1)=0.0D0
      xmoi(3,2)=0.0D0
      xmoi(3,3)=193125.0D0*.9          ! IZZ=.9*Iyy
      call inv3(xmoi,xmoi_inv)         ! inverse matrix
c
      rcm(1)=1.0D0                     ! center of mass location in geometric body
      rcm(2)=0.1D0                     ! coordinates, meters
      rcm(3)=0.1D0
c
c *** Control system gains:
c
      a0=1000.0D0                      ! control gain, x-axis position, n-m/rad
      a1=100.0D0                       ! control gain, x-axis rate, n-m-sec/rad
      b0=1000.0D0                      ! y-axis
      b1=100.0D0
      c0=1000.0D0                      ! z-axis
      c1=100.0D0
c
      print*,'    Input stepsize reduction factor'
      print*,'    for ATTITUDE DYNAMICS simulation'
      print*,'    (integer):'
      read*,idiv
      return
c
c ************************************************************************************************
```

```
c ******************************** End of mode 1, beginning of mode 2 ********************************
c ****************************************************************************************************
c
  200 if(mode.eq.3) go to 300
      continue ! (mode=2, set initial conditions)
c
c ****************************************************************************************************
c ***************************** Setting Initial Conditions (mode 2 function): ****************************
c ****************************************************************************************************
c
      phi(1)=0.0D0            ! initial values of the Euler angles (radians)
      phi(2)=0.0D0
      phi(3)=0.0D0
c
      omega(1)=0.0D0          ! initial values of the body rates (radians)
      omega(2)=0.0D0
      omega(3)=0.0D0
c
      do 5 i=1,3              !entering initial conditions into block common /endcond2/ (radians)
      yy2i(i)=phi(i)
      yy2i(i+3)=omega(i)
    5 continue
c
      return
c
c ****************************************************************************************************
c ****************************************************************************************************
c ******************************** End of mode 2, beginning of mode 3 ********************************
c ****************************************************************************************************
c ****************************************************************************************************
c
  300 continue ! (mode=3, run simulation over the convergence interval)
c
c ****************************************************************************************************
c ******************** Setting up the initial state vector (6 states, mode 2 function): ****************
c ****************************************************************************************************
c
      do 6 i=1,3              !retrieving initial conditions from block common /endcond2/ (radians)
```

```
      phi(i)=yy2i(i)
      omega(i)=yy2i(i+3)
c
      yy2(i,1)=phi(i)        !entering initial output vector (time point number 1) into common/subsys2/
      yy2(i+3,1)=omega(i)
c
    6 continue

c
      do 10 i=1,3            ! changing from kinematic to state variable notation
      x(i)=phi(i)            ! states 1-3, Euler angles, radians
      x(i+3)=omega(i)        ! states 4-6, body rates, rad/sec
   10 continue
c      print*,'from att.f******'
c      print*,'uu2',(uu2(1,j),j=1,ndp)
c      print*
c      print*,'    ',(uu2(2,j),j=1,ndp)
c      print*
c      print*,'    ',(uu2(3,j),j=1,ndp)
c      print*
c
      tau=0.0D0 !initializing time, measured from the start of the convergence interval            !
c
      ndim=6 ! defining dimension of the state vector
c
      continue !******************** begin integration loop ********************************************
c
      do 20 i=2,ndp          ! generates points 2 through ndp
c
      dt=delt/float(idiv)
      do 30 ii=1,idiv
      call rksys2(tau,x,ndim,dt) ! propagates attitude states across convergence interval
   30 continue
c
      do 7 j=1,6                    ! enters output (Euler angles and body rates) into common/susbsy2/
      yy2(j,i)=x(j)
    7 continue
c
```

```
   20 continue
c
c ********************************** end integration loop ***********************************************
c
      return
c
c ***************************************************************************************************************
c ***************************************************************************************************************
c ************************************* End of mode 3 ***********************************************
c ***************************************************************************************************************
c ***************************************************************************************************************
c
      end

xxxx
```

```
      implicit double precision(a-h,o-z)
c
      dimension yy(18,100),yyp(18,100)
      dimension ynew(100),yold(100)
c
      common/simulation/ndp,delt,tol,res,t_start,t_report,t_print,
     1t_stop,nmt,nmtr,nit,t_initial,t_final
c
c ***************************************************************************************************
c ************************************* begin check loop ********************************************
c ***************************************************************************************************
c
      iconflag=0          !initializes system-level flag to "converged"

      do 20 i=1,nsysdim   !(nsysdim=18). Loads current and previous scalar signals for
      do 19 j=1,ndp       !comparison (convergence check).
c
      ynew(j)=yy(i,j)
      yold(j)=yyp(i,j)
c
   19 continue
c
      call convrg(ynew,yold,ndp,tol,res,iflag) ! iflag=1 for non-convergence.
c
      iconflag=iconflag+iflag !computes status of system-level convergence
c
   20 continue
c
      if(iconflag.gt.0) iconflag=1 !routine returns 0 or 1 for iconflag
c
c ***************************************************************************************************
c ************************************** end check loop *********************************************
c ***************************************************************************************************
c
      return

c
      end
```

```
xxxx
```

```
      IF(YNEW(I).LT.YMIN) YMIN=YNEW(I)
   11 CONTINUE
c
      SPAN=dABS(YMAX-YMIN)                         !6/22/00
      CHECK2=TOL*SPAN
c
      IF(CHECK1.LE.CHECK2.OR.CHECK1.LE.RES) IFLAG=0
c
      RETURN

      END

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx


      subroutine coupler(yy,uu)
c
      implicit double precision(a-h,o-z)
c
c
      dimension uu(18,100),yy(18,100) !system-level I/O, 18 input and 18 output signals, 100 time points
c
      dimension f_aero(3)                !aerodynamic force (drag), inertial coordinates
      dimension f_aero_b(3), t_aero_b(3) !aerodynamic force and torque, body frame
      dimension f_aero_g(3), t_aero_g(3) !aerodynamic force and torque, geometric frame
      dimension rxf(3)                   !aerodynamic force transfer couple (from geometric origin to c.m.)
      dimension rad(3),vel(3)            !orbit radius and velocity vectors, inertial coordinates
      dimension rad_b(3),vel_b(3)        !orbit radius and velocity vectors, body coordinates
      dimension phi(3)                   !Euler angles, inertial frame to body frame, 3-2-1 sequence
      dimension amat(3,3)                !direction cosine matrix (d.c.m.), inertial to body
      dimension amat_t(3,3)              !transpose of amat(i,j)
      dimension xmoi(3,3),xmoi_inv(3,3)  !inertia matrix, inverse inertia matrix
      dimension rcm(3)                   !center of mass location, geometric coordinates
```

```
c
      common/simulation/ndp,delt,tol,res,t_start,     !all except some single-time-point utility routines
     1t_report,t_print,t_stop,nmt,nmtr,nit,
     2t_initial,t_final
c
      common/mass/xmass,xmoi,xmoi_inv,rcm            !orbit.f, sub1.f, attitude.f, sub2.f, coupler.f
c
c ***************************************************************************************************
c ********************************* Coupling Aerodynamics/Orbit Dynamics ********************************
c ***************************************************************************************************
c
      do 10 j=1,ndp
c
c *** from aerodynamics to orbit dynamics:
c
      f_aero_b(1)=yy(13,j)      !extracting aerodynamic (drag) force from system output signal
      f_aero_b(2)=yy(14,j)      !computed to act at origin of geometric body frame,
      f_aero_b(3)=yy(15,j)      !but can be assummed to act at c.m. for this purpose
c
      phi(1)=yy(7,j)            !extracting attitude (Euler) angles from the system output signal
      phi(2)=yy(8,j)
      phi(3)=yy(9,j)
c
      call dcm321(phi,amat)     !computes d.c.m that transforms vectors from inertial to body coordinates
      call tran3(amat,amat_t)   !computes d.c.m that transforms vectors from body to inertial coordinates
c
      call matvec(amat_t,f_aero_b,f_aero) !transforms force vector from body to inertial coordinates
c
      uu(1,j)=f_aero(1)     !feedback signal is aerodynamic force (drag) in inertial coordinates,
      uu(2,j)=f_aero(2)     !and is assummed to act at the satellite center of mass (c.m.)
      uu(3,j)=f_aero(3)
c
      uu(4,j)=0.0D0         !slot holders, nothing yet identified
      uu(5,j)=0.0D0
      uu(6,j)=0.0D0
c
c *** from orbit dynamics to attitude dynamics:
c
```

```
      rad(1)=yy(1,j)          !Extracting orbit radius from the system output signal
      rad(2)=yy(2,j)          !variable rad(i) is used for listing readability only
      rad(3)=yy(3,j)
c
      uu(7,j)=rad(1)          !feedback signal is orbit radius vector in inertial coordinates
      uu(8,j)=rad(2)
      uu(9,j)=rad(3)
c
c *** from aerodynamics to attitude dynamics:
c
      f_aero_g(1)=yy(13,j)    !aerodynamic (drag) force; acts at origin of geometric body frame.
      f_aero_g(2)=yy(14,j)    !f_aero_g(i) and f_aero_b(i) are numerically indentical, but their
      f_aero_g(3)=yy(15,j)    !physical interpretations (points of action) may be different
c
      t_aero_g(1)=yy(16,j)    !aerodynamic torque with force at origin of geometric body frame
      t_aero_g(2)=yy(17,j)
      t_aero_g(3)=yy(18,j)
c
      call vcross(rcm,f_aero_g,rxf)          ! torque of drag force about c.m.
c
      t_aero_b(1)=t_aero_g(1)+rxf(1)         ! transferring drag force to c.m. and adding an
      t_aero_b(2)=t_aero_g(2)+rxf(2)         ! eqivalent couple.
      t_aero_b(3)=t_aero_g(3)+rxf(3)
c
      uu(10,j)=t_aero_b(1)
      uu(11,j)=t_aero_b(2)
      uu(12,j)=t_aero_b(3)
c
c *** from orbit dynamics to aerodynamics
c
      call matvec(amat,rad,rad_b) !transforms orbit radius vector to body coordinates
c
      uu(13,j)=rad_b(1)           !feedback signal is orbit radius vector in body coordinates
      uu(14,j)=rad_b(2)
      uu(15,j)=rad_b(3)
c
c
      vel(1)=yy(4,j)              !variable vel(i) is used for listing readability only
```

```
      vel(2)=yy(5,j)
      vel(3)=yy(6,j)
c
      call matvec(amat,vel,vel_b) !transforms orbit velocity vector to body coordinates

      uu(16,j)=vel_b(1)           !feedback signal is orbit velocity vector in body coordinates
      uu(17,j)=vel_b(2)
      uu(18,j)=vel_b(3)
c
   10 continue
c
      return
      end

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      subroutine dcm2ang321(amat,phi)
c
c *** computes equivalent 3-2-1 sequence Euler angles for d.c.m amat(i,j)
c
      implicit double precision (a-h,o-z)
c
      dimension amat(3,3), phi(3)
c
      phi(1)=datan2(amat(2,3),amat(3,3))      ! (+/- pi   rad.)
      phi(2)=-dasin(amat(1,3))                ! (+/- pi/2 rad.)
      phi(3)=datan2(amat(1,2),amat(1,1))      ! (+/- pi   rad.)
c
      return
      end
```

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx


      subroutine dcm321(phi,amat)
c
c *** Created 5/12/2000 to support "standard.f/sub.f". constructs
c     a DCM (direction cosine matrix) from the Euler angles phi(i).
c     The rotation sequence is 3-2-1, and the resulting DCM (amat)
c     transforms vector components FROM the original cordinate frame
c     (inertial) TO the final (body) cordinate frame.
c
      implicit double precision (a-h,o-z)
c
      dimension phi(3),amat(3,3),ax(3,3),ay(3,3),az(3,3)
c
      cx=dcos(phi(1))
      sx=dsin(phi(1))
      cy=dcos(phi(2))
      sy=dsin(phi(2))
      cz=dcos(phi(3))
      sz=dsin(phi(3))
c
      do 1 i=1,3
      do 1 j=1,3
      ax(i,j)=0.0D0
      ay(i,j)=0.0D0
      az(i,j)=0.0D0
    1 continue
c
      ax(1,1)=1.0D0
      ax(2,2)=cx
      ax(2,3)=sx
      ax(3,2)=-sx
```

```
      ax(3,3)=cx
c
      ay(2,2)=1.0D0
      ay(1,1)=cy
      ay(1,3)=-sy
      ay(3,1)=sy
      ay(3,3)=cy
c
      az(3,3)=1.0D0
      az(1,1)=cz
      az(1,2)=sz
      az(2,1)=-sz
      az(2,2)=cz
c
      do 2 i=1,3
      do 2 l=1,3
      amat(i,l)=0.0D0
      do 2 k=1,3
      do 2 j=1,3
      amat(i,l)=amat(i,l)+ax(i,j)*ay(j,k)*az(k,l)
    2 continue
c
      return
      end

xxxx
```

```
c
      dimension xx(18,100),xx1(6,100),xx2(6,100),xx3(6,100)
c
c     Assumes nsubdim number of scalar signals per subsystem
c
      do 10 i=1,nsubdim              !eighteen scaler signals at the system level,
      do 10 j=1,ndp                  !ndp time points each
      xx1(i,j)=xx(i,j)
      xx2(i,j)=xx(i+6,j)
      xx3(i,j)=xx(I+12,j)
   10 continue
c
      return
      end

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx


      subroutine interp(ndp,tstart,delt,time,func,val)
c
c *** Linearly interpolates func(t) data for a particular value of time.
c
      implicit double precision(a-h,o-z)
c
      dimension func(100) !single scalar signal, 100 time points
      interval=1
      t1=tstart
c
    1 if(time.lt.t1+delt) go to 3
c
    2 continue              ! continuing search for the interval
      interval =interval +1
      t1=t1+delt
```

```
      go to 1
c
    3 continue                 ! performing the interpolation
c
      i=interval
      fac=(time-t1)/delt
      val=func(i)+fac*(func(i+1)-func(i))
c
      return
      end

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx


      SUBROUTINE INV3(A,AI)
C
C *** THIS ROUTINE TAKES THE INVERSE OF A 3X3 MATRIX.
C
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C
      DIMENSION A(3,3),AI(3,3),TEMP(3,3)
C
      DET=A(1,1)*(A(2,2)*A(3,3)-A(3,2)*A(2,3))
     C-A(2,1)*(A(1,2)*A(3,3)-A(3,2)*A(1,3))
     C+A(3,1)*(A(1,2)*A(2,3)-A(2,2)*A(1,3))
      TEMP(1,1)=A(2,2)*A(3,3)-A(3,2)*A(2,3)
      TEMP(1,2)=A(3,1)*A(2,3)-A(2,1)*A(3,3)
      TEMP(1,3)=A(2,1)*A(3,2)-A(3,1)*A(2,2)
      TEMP(2,1)=A(3,2)*A(1,3)-A(1,2)*A(3,3)
      TEMP(2,2)=A(1,1)*A(3,3)-A(3,1)*A(1,3)
      TEMP(2,3)=A(3,1)*A(1,2)-A(1,1)*A(3,2)
      TEMP(3,1)=A(1,2)*A(2,3)-A(2,2)*A(1,3)
      TEMP(3,2)=A(2,1)*A(1,3)-A(1,1)*A(2,3)
```

```
      TEMP(3,3)=A(1,1)*A(2,2)-A(2,1)*A(1,2)
      DO 10 I=1,3
      DO 10 J=1,3
   10 AI(I,J)=TEMP(J,I)/DET
      RETURN
      END

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx


      SUBROUTINE MATVEC(A,V,AV)
C
C *** THIS ROUTINE MULTIPLIES A VECTOR BY A MATRIX.
C     (3X3)*(3X1)=(3X1)
C
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C
      DIMENSION A(3,3),V(3),AV(3)
C
      DO 10 I=1,3
      AV(I)=0.0
      DO 10 J=1,3
   10 AV(I)=AV(I)+A(I,J)*V(J)
      RETURN
      END

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
      subroutine orbit(mode) ! mode 1= set physical constants, system parameters,etc.
                             ! mode 2= set initial conditions
                             ! mode 3= run simulation over the convergence interval
c
c *** This is subprogram "orbit.f". Construction began 6/19/2000 by Doug Nixon, TD55, MSFC.
c    "orbit.f" is the main driver for the six-state dynamic system which is integrated
c     by "rksys.f", a fourth-order Runge-Kutta subroutine. "sub1.f" defines the
c     dynamics in first-order state-variable form.
c
      implicit double precision (a-h,o-z)
c
      dimension uu1(6,100),yy1(6,100)
      dimension yy1i(6),yy1f(6)

      dimension r(3),v(3)
      dimension x(12),xdot(12)
      dimension xmoi(3,3),xmoi_inv(3,3)  !inertia matrix, inverse inertia matrix
      dimension rcm(3)                   !center of mass location, geometric coordinates
c
      common/subsys1/uu1,yy1        !orbit dynamics, input/output vs. time
      common/endcond1/yy1i,yy1f     !initial and final output conditions
c
      common/simulation/ndp,delt,tol,res,t_start,     !all except some single-time-point utility routines
     1t_report,t_print,t_stop,nmt,nmtr,nit,
     2t_initial,t_final
c
      common/constants/rearth,gme,pi,cvdr             !recursive.f, orbit.f,sub1, attitude.f, sub2.f
c
      common/mass/xmass,xmoi,xmoi_inv,rcm            !orbit.f, sub1.f, attitude.f, sub2.f, coupler.f
c
      if(mode.eq.2.or.mode.eq.3) go to 200
c
  100 continue ! (mode=1, define constants, system parameters)
c
c ********************************************************************************************
c ************************** Numerically Defining Constants (mode 1 function): ***************
c ********************************************************************************************
```

```
c
      gravconst=(6.673D0)*(1.0e-11)    ! Constant of Gravitation,
                                       ! m**3/kg-sec**2, CRC, page 6
c
      rearth= 6367.65D0                ! Earth's radius,(polar+equatorial)/2.0,
                                       ! km, CRC, page 6
c
      dearth=5522.0D0                  ! Average density of the earth,
                                       ! kg/m**3, CRC, page 6
      pi=datan2(0.D0,-1.D0)            ! Computes pi to machine precision
      cvdr=pi/180.0D0                  ! Converts degrees to radians
c
      volume=(4.0D0/3.0D0)*pi*(rearth*1000.0D0)**3  ! Volume of earth, m**3
c
      xmearth=dearth*volume            ! Mass of the earth, kg
c
      gme=gravconst*xmearth            ! Gravity Constant times mass of the earth.
c
c ***************************************************************************************
c ********************* Setting Values of System Parameters (mode 1 function): **********
c ***************************************************************************************
c
      xmass=10000.0D0                  ! spacecraft mass, kilograms
c
      print*,'    Input stepsize reduction factor'
      print*,'    for ORBIT DYNAMICS simulation'
      print*,'    (integer):'
      read*,idiv
      return
c
c ***************************************************************************************
c ***************************************************************************************
c ********************************* End of mode 1, beginning of mode 2 ******************
c ***************************************************************************************
c ***************************************************************************************

  200 if(mode.eq.3) go to 300
      continue ! (mode=2, set initial conditions)
```

```
c
c ***************************************************************************************************
c ************************** Setting Initial Conditions (mode 2 function): **************************
c ***************************************************************************************************
c
      alt=300.0D0                          ! altitude, kilometers
c
      rmag=(rearth+alt)*1000.0D0           ! meters
      r(1)=rmag*dcos(30.0d0*cvdr)
      r(2)=0.0d0                           ! 30 deg "beta angle", (between ecliptic
      r(3)=rmag*dsin(30.0d0*cvdr)          ! and orbit planes)
c
      vmag=dsqrt(gme/rmag)                 ! Circular orbit, meters/sec
      rcube=rmag**3
      wsq=gme/rcube
      cperiod=2.0d0*pi/dsqrt(wsq)          ! Period for circular orbit
      v(1)=0.0
      v(2)=vmag
      v(3)=0.0
c
      do 5 i=1,3              !entering initial conditions into block common /endcond1/
      yy1i(i)=r(i)
      yy1i(i+3)=v(i)
    5 continue
c
      return
c
c ***************************************************************************************************
c ***************************************************************************************************
c ********************************* End of mode 2, beginning of mode 3 ******************************
c ***************************************************************************************************
c ***************************************************************************************************
c
  300 continue ! (mode=3, run simulation over the convergence interval)
c
c ***************************************************************************************************
c ******************* Setting up the initial state vector (6 states, mode 2 function): **************
c ***************************************************************************************************
```

```
c
      do 6 i=1,3              !retrieving initial conditions from block common /endcond1/
      r(i)=yy1i(i)
      v(i)=yy1i(i+3)
c
      yy1(i,1)=r(i)           !entering initial output vector (time point number 1) into block
      yy1(i+3,1)=v(i)         !common /subsys1/
c
    6 continue

c
      do 10 i=1,3             ! changing from kinematic to state variable notation
      x(i)=r(i)               ! states 1-3, orbit radius, meters
      x(i+3)=v(i)             ! states 4-6, orbit velocity, meters/sec
   10 continue
c
      tau=0.0D0 !initializing time, measured from the start of the convergence interval          !
c
      ndim=6 ! defining dimension of the state vector
c
      continue !******************** begin integration loop ********************************************
c
      do 20 i=2,ndp
c
      dt=delt/float(idiv)         ! smaller stepsize than delt
      do 30 ii=1,idiv
      call rksys1(tau,x,ndim,dt)  ! propagates orbit states across convergence interval
   30 continue
c
      do 7 j=1,6                    ! enters output (orbit state vector) into block common /susbsy1/.
      yy1(j,i)=x(j)                 ! data (time) points 2 through ndp are dealt with here
    7 continue
c
   20 continue
c
c ********************************** end integration loop ********************************************
c
      return
```

```
c
c ***************************************************************************************************
c ***************************************************************************************************
c ***************************************************************************************************
c ********************************** End of mode 3 **************************************************
c ***************************************************************************************************
c ***************************************************************************************************
c
      end

xxxx
```

```
      common/constants/rearth,gme,pi,cvdr          !recursive.f, orbit.f,sub1, attitude.f, sub2.f
c
      common/mass/xmass,xmoi,xmoi_inv,rcm             !orbit.f, sub1.f, attitude.f, sub2.f, coupler.f
c                                                     !added 9/28/2000, for aero torque output
c
      do 3 i=1,18
      do 3 j=1,ndp
      zz(i,j)=yy(i,j)
    3 continue
c
      do 4 i=1,3
      do 4 j=1,ndp
      phi(i)=zz(i+6,j)                ! unlimited angle (10/16/00)
      call dcm321(phi,amat)
      call dcm2ang321(amat,phi)
      zz(i+6,j)=phi(i)/cvdr           ! limited angle   (10/16/00)
      zz(i+9,j)=yy(i+9,j)/cvdr
c
    4 continue
c
      do 5 j=1,ndp                    !added 9/28/2000 for aero. torque compution for output
      f_aero(1)=zz(13,j)
      f_aero(2)=zz(14,j)
      f_aero(3)=zz(15,j)
      call vcross(rcm,f_aero,t_aero)
      zz(16,j)=t_aero(1)
      zz(17,j)=t_aero(2)
      zz(18,j)=t_aero(3)
    5 continue
c
      time1=nmt*(ndp-1)*delt    !nmt=number of major time (convergence) intervals complete
c                               !ndp=number of data points per major interval, including both end points
c                               !delt=time between data points (also integration stepsize)
      time=time1                !initializing time to first point of convergence interval
c
      do 1000 i=1,ndp-1                        ! begin first write loop
c
        do 1 k=1,3                                  !post processing orbit data for altitude
```

```
          rad(k)=yy(k,i)
      1   continue
          call vecnorm(rad,rad_n,rmag)
          alt=rmag/1000.0D0-rearth  !altitude, kilometers
c
          if(time.ge.t_print) write(1,100) time,(zz(j,i),j=1,18),alt,nit
          if(time.ge.t_print) t_print=t_print+del_print
c
          if(time.ge.t_report) ipd=int((time/t_stop)*100.0)
          if(time.ge.t_report) print*,'Percent complete = ',
       1  ipd
          if(time.ge.t_report) t_report=t_report+del_report
    100 format(20E20.10,i5)
        time=time+delt
   1000 continue
c
c       time=time1                !re-nitializing time to first point of convergence interval
c
c       do 2000 i=1,ndp-1                          ! begin second write loop
c
c       write(2,100) time,(uu(j,i),j=1,18) !system-level feedback signal
c    200 format(19E20.10)
c       time=time+delt
c  2000 continue
c
        return
        end

xxxx
```

```
C
C     THIS SUBROUTINE PERFORMS ONE STEP IN THE RUNGE-KUTTA
C     INTEGRATION OF A SYSTEM OF FIRST ORDER DIFFERENTIAL
C     EQUATIONS OF THE FORM Y'(X)=FD(X,Y), WHERE Y AND FD
C     ARE VECTOR FUNCTIONS, AND X IS THE INDEPENDENT
C     VARIABLE.  THE ALGORITHM IS DESCRIBED BY THE FOLLOWING
C     VECTOR RELATIONS:
C
C          F1=FD(X,Y),              YT=Y+F1*H/6
C          F2=FD(X+H/2,Y+F1*H/2),   YT=YT+F2*H/3
C          F3=FD(X+H/2,Y+F2*H/2),   YT=YT+F3*H/3
C          F4=FD(X+H,Y+F3*H),        Y=YT+F4*H/6
C           X=X+H,                  NEXT STEP
C
      DIMENSION Y(6),YSTART(6),YTEMP(6),FDERIV(6) ! Set for point mass orbit dynamics
      H2=H/2.0
      H3=H/3.0
      H6=H/6.0
      X0=X
      DO 200 I=1,N
  200 YSTART(I)=Y(I)
      CALL SUB1(FDERIV,Y,X)          ! "1" added to identify with subsystem #1
      DO 300 I=1,N
      YTEMP(I)=YSTART(I)+FDERIV(I)*H6
  300 Y(I)=YSTART(I)+FDERIV(I)*H2
C
C *** END OF FIRST PASS ***
C
      X=X0+H2
      CALL SUB1(FDERIV,Y,X)          ! "1" added to identify with subsystem #1
      DO 400 I=1,N
      YTEMP(I)=YTEMP(I)+FDERIV(I)*H3
  400 Y(I)=YSTART(I)+FDERIV(I)*H2
C
C *** END OF SECOND PASS ***
C
      CALL SUB1(FDERIV,Y,X)          ! "1" added to identify with subsystem #1
      DO 500 I=1,N
```

```
      YTEMP(I)=YTEMP(I)+FDERIV(I)*H3
  500 Y(I)=YSTART(I)+FDERIV(I)*H
C
C *** END OF THIRD PASS ***
C
      X=X0+H
      CALL SUB1(FDERIV,Y,X)          ! "1" added to identify with subsystem #1
      DO 600 I=1,N
  600 Y(I)=YTEMP(I)+FDERIV(I)*H6
C
C *** END OF FOURTH PASS ***
C
C     AT THIS POINT ONE INTEGRATION STEP HAS BEEN COMPLETED.
C     Y(X) HAS BEEN REPLACED BY Y(X+H), AND
C     X HAS BEEN REPLACED BY X+H.
C
      RETURN
      END

Xxxx
```

```
C           F1=FD(X,Y),              YT=Y+F1*H/6
C           F2=FD(X+H/2,Y+F1*H/2),   YT=YT+F2*H/3
C           F3=FD(X+H/2,Y+F2*H/2),   YT=YT+F3*H/3
C           F4=FD(X+H,Y+F3*H),   '    Y=YT+F4*H/6
C            X=X+H,                  NEXT STEP
C
      DIMENSION Y(6),YSTART(6),YTEMP(6),FDERIV(6) ! Set for rigid body rotational dynamics
      H2=H/2.0
      H3=H/3.0
      H6=H/6.0
      X0=X
      DO 200 I=1,N
  200 YSTART(I)=Y(I)
      CALL SUB2(FDERIV,Y,X)          ! "2" added to identify with subsystem #2
      DO 300 I=1,N
      YTEMP(I)=YSTART(I)+FDERIV(I)*H6
  300 Y(I)=YSTART(I)+FDERIV(I)*H2
C
C *** END OF FIRST PASS ***
C
      X=X0+H2
      CALL SUB2(FDERIV,Y,X)          ! "2" added to identify with subsystem #2
      DO 400 I=1,N
      YTEMP(I)=YTEMP(I)+FDERIV(I)*H3
  400 Y(I)=YSTART(I)+FDERIV(I)*H2
C
C *** END OF SECOND PASS ***
C
      CALL SUB2(FDERIV,Y,X)          ! "2" added to identify with subsystem #2
      DO 500 I=1,N
      YTEMP(I)=YTEMP(I)+FDERIV(I)*H3
  500 Y(I)=YSTART(I)+FDERIV(I)*H
C
C *** END OF THIRD PASS ***
C
      X=X0+H
      CALL SUB2(FDERIV,Y,X)          ! "2" added to identify with subsystem #2
      DO 600 I=1,N
```

```
  600 Y(I)=YTEMP(I)+FDERIV(I)*H6
C
C *** END OF FOURTH PASS ***
C
C     AT THIS POINT ONE INTEGRATION STEP HAS BEEN COMPLETED.
C     Y(X) HAS BEEN REPLACED BY Y(X+H), AND
C     X HAS BEEN REPLACED BY X+H.
C
      RETURN
      END

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx


c
      subroutine sub1(xdot,x,tau)
c
c *** This subroutine simulates three-degree-of-freedom (six-state) orbital dynamics with inverse
c     square gravity and point mass assumptions. An aerodynamic drag force f_aero(i) can be applied
c     to the satellite through common block "subsys1" variable uu1(i,j), where the following
c     definitions apply:
c
c                          x(i) = value of the ith component of the orbit state vector. This is an
c                                 INPUT VARIABLE, and is not changed by this routine.
c
c                       xdot(i) = time derivative of the ith component of the orbit state vector.
c                                 This is an OUTPUT VARIABLE, and is created bythis routine.
c
c                           tau = time elapsed from the beginning of the convergence interval. This
c                                 is an INPUT VARIABLE, and is not changed.
c
c     .                uu1(1,j) = value at the jth time point of the x-component of body force f(t)
c                                 expressed in inertial coordinates. Units are newtons. This is an
```

```
c                          INPUT VARIABLE, and is not changed.
c
c                 uu1(2,j) = value at the jth time point of the y-component of body force f(t)
c                            expressed in inertial coordinates. Units are newtons. This is an
c                            INPUT VARIABLE, and is not changed.
c
c                 uu1(3,j) = value at the jth time point of the z-component of body force f(t)
c                            expressed in inertial coordinates. Units are newtons. This is an
c                            INPUT VARIABLE and is not changed.
c
c                 yy1(i,j) = value at the jth time point of the ith component of the orbit state
c                            vector. THIS VARIABLE IS NOT USED BY THIS ROUTINE.
c
c                      ndp = number of signal sample points per convergence interval, including
c                             both end points
c
c                     delt = time between signal sample points, same as integration stepsize.
c
c
      implicit double precision(a-h,o-z)
c
      dimension uu1(6,100),yy1(6,100)
c
      dimension f_aero(3)        !aerodynamic force, single time point, inertial coordinates
      dimension f_grav(3)        !gravitational force, single time point, inertial coordinates
      dimension rnorm(3)         !normalized radius vector, inertial coordinates
c
      dimension func_x(100),func_y(100),func_z(100) ! vector input signal, 100 time points
c                                                   ! (aerodynamic force)
c
      dimension x(6),xdot(6),r(3),rdot(3),v(3),vdot(3)
c
      dimension xmoi(3,3),xmoi_inv(3,3)  !inertia matrix, inverse inertia matrix
c
      dimension rcm(3)                   !center of mass location, geometric coordinates
c
      common/subsys1/uu1,yy1        !orbit dynamics, input/output vs. time
      common/endcond1/yy1i,yy1f     !initial and final output conditions
```

```
c
      common/simulation/ndp,delt,tol,res,t_start,     !all except some single-time-point utility routines
     1t_report,t_print,t_stop,nmt,nmtr,nit,
     2t_initial,t_final
c
      common/constants/rearth,gme,pi,cvdr             !recursive.f, orbit.f,sub1, attitude.f, sub2.f
c
      common/mass/xmass,xmoi,xmoi_inv,rcm            !orbit.f, sub1.f, attitude.f, sub2.f, coupler.f
c
c ***************************************************************************************************
c
c                 Orbit States (Vehicle position and velocity, inertial coordinates):
c
c                 x(1)=r(1): x-component of orbit radius vector (meters)
c                 x(2)=r(2): y-component
c                 x(3)=r(3): z-component
c
c                 x(4)=v(1): x-component of spacecraft velocity vector, (meters/second)
c                 x(5)=v(2): y-component
c                 x(6)=v(3): z-component
c
c
c ***************************************************************************************************
c
      r(1)=x(1)         ! Changing from mathematical (state variable) to kinematic notation.
      r(2)=x(2)
      r(3)=x(3)
      v(1)=x(4)
      v(2)=x(5)
      v(3)=x(6)
c
c *** Gravity forces ***
c
      call vecnorm(r,rnorm,rmag)       ! normalized radius vector (inertial coordinates)
      gmag=gme*xmass/rmag**2
c
      f_grav(1)=-gmag*rnorm(1)         ! result in inertial coordinates
      f_grav(2)=-gmag*rnorm(2)
```

```
      f_grav(3)=-gmag*rnorm(3)
c
c *** Aerodynamic forces ***
c
      t1=0.0D0 ! time (tau) is measured fro the left end point of the convergence interval
c
      do 1 i=1,ndp          !extracts scaler components of the input vector signal form the
      func_x(i)=uu1(1,i)    !system-level common block "subsys1". The variable uu1(i,j) contains
      func_y(i)=uu1(2,i)    !aerodynamic forces as described above.
      func_z(i)=uu1(3,i)
    1 continue
c
      call interp(ndp,t1,delt,tau,func_x,val)  !interpolates x-component of aerodynamic force
      f_aero(1)=val
      call interp(ndp,t1,delt,tau,func_y,val)  !interpolates y-component of aerodynamic force
      f_aero(2)=val
      call interp(ndp,t1,delt,tau,func_z,val)  !interpolates z-component of aerodynamic force
      f_aero(3)=val
ccccc      print*,'sub1,after interp',(f_aero(i),i=1,3)
c
c *** orbital dynamics (Orbit State Derivatives) ***
c
      vdot(1)=(1.0D0/xmass)*(f_grav(1)+f_aero(1))   !acceleration, inertial coordinates
      vdot(2)=(1.0D0/xmass)*(f_grav(2)+f_aero(2))
      vdot(3)=(1.0D0/xmass)*(f_grav(3)+f_aero(3))
c
      rdot(1)=v(1)                                  !velocity, inertial coordinates
      rdot(2)=v(2)
      rdot(3)=v(3)
c
c ***************************************************************************************
c
      xdot(1)=rdot(1) !changing back to mathematical notation from physical notation
      xdot(2)=rdot(2)
      xdot(3)=rdot(3)
      xdot(4)=vdot(1)
      xdot(5)=vdot(2)
      xdot(6)=vdot(3)
```

```
c
      return
      end

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      subroutine sub2(xdot,x,tau)
c
c *** Construction began 6/28/2000, by Doug Nixon, TD55. This is the attitude dynamics (subsystem 2)
c     routine driven by rksys2.f which supports attitude.f. Attitude dynamics is is the 2nd subsystem
c     of three (ordit dynamics, attitude dynamics, and aerodynamics) that form a system simulation
c     implemented by recursive feedback.
c
c
      implicit double precision (a-h,o-z)
c
      dimension uu2(6,100),yy2(6,100)                 !I/O for subsystem 2 (attitude dynamics)
c
      dimension x(6),xdot(6)                          !state variables for attitude dynamics
c
      dimension phi(3),phidot(3),omega(3),omegadot(3) !attitude angles and body rates (state variables)
c
      dimension rad(3),rad_norm(3),rad_norm_b(3)      !orbit radius vector (meters)
c
      dimension xmoi(3,3),xmoi_inv(3,3),rcm(3)        !moments of inertia, about c.m, c.m. location vector
c                                                     !in geometric body coordinates
c
      dimension t_gg(3)                               !gravity gradient torque (body coordinates)
c
      dimension t_aero(3)                             !aerodynamic torque (body coordinates)
c
      dimension t_control(3)                          !control torque (body coordinates)
```

```
c
      dimension t_total(3)                                !total external torque minus gyroscopic torque
c
      dimension cmat(3,3)                                 !transforms body rates to Euler angle rates
c
      dimension amat(3,3),amat_t(3,3)                     !transforms vectors from inertial to body coordinates
c
      dimension h(3), wxh(3)                              !angular momentum about c.m., gyroscopic torque
c
      dimension xir(3), rxxir(3)                          ! [ I]{ r} , { r} X[ I]{ r} , I=xmoi(i,j), r=rad_norm_b(i)
c
      dimension func_x(100), func_y(100), func_z(100)  !scalar version of aerodynamic torque input signal
c
      common/subsys2/uu2,yy2                              !recursive.f, attitude.f, sub2.f
c
      common/simulation/ndp,delt,tol,res,t_start,         !all except some single-time-point utility routines
     1t_report,t_print,t_stop,nmt,nmtr,nit,
     2t_initial,t_final
c
      common/constants/rearth,gme,pi,cvdr                 !recursive.f, orbit.f,sub1, attitude.f, sub2.f
c
      common/mass/xmass,xmoi,xmoi_inv,rcm                !orbit.f, sub1.f, attitude.f, sub2.f, coupler.f
c
      common/gains/a0,a1,b0,b1,c0,c1                      !attitude.f, sub2.f
c
c *** Some variable in definitions:
c
c       gme  : universal gravitational constant times mass of the earth (lb-ft**2/slug)
c       xmass: spacecraft mass (slugs)
c       xmoi : spacecraft moment of inertia tensor (3x3 matrix, slug-ft**2)
c
c     The system has three degrees of freedom, or six states. The System states are
c     defined as follows:
c
c
c     Vehicle Rotational States:
c
c       x(1)=phi(1): Euler angle rotation from inertial to body system, about x-axis,
```

```
c                    3rd in sequence. (radians)
c       x(2)=phi(2): About y-axis, 2nd in sequence.
c       x(3)=phi(3): About z-axis, 1st in sequence.
c
c       x(4)=omega(1):  Body rate, x-axis (radians/second)
c       x(5)=omega(2):  Body rate, x-axis
c       x(6)=omega(3):  Body rate, x-axis
c
c ***************************************************************************
c


      phi(1)=x(1)                ! Changing from mathematical to physical notation.
      phi(2)=x(2)
      phi(3)=x(3)
c
      omega(1)=x(4)
      omega(2)=x(5)
      omega(3)=x(6)
c
      call dcm321(phi,amat)     !computes transformation (DCM) from inertial
c                               !to body coordinates
c
      call tran3(amat,amat_t)   !transformation from body to inertial coordinates
c
      do 1 i=1,ndp           !extracts scaler components of the input vector signal from the
      func_x(i)=uu2(1,i)     !common block "subsys2". The variable uu2(1-3,j) contains the orbit
      func_y(i)=uu2(2,i)     !radius vector inertial coordinates.
      func_z(i)=uu2(3,i)
    1 continue
c
      t1=0.0D0 ! time (tau) is measured from the left end point of the convergence interval
c
      call interp(ndp,t1,delt,tau,func_x,val)  !interpolates x-component of the radius vector
      rad(1)=val
      call interp(ndp,t1,delt,tau,func_y,val)  !... y-component ...
      rad(2)=val
      call interp(ndp,t1,delt,tau,func_z,val)  !... z-component ...
```

```
      rad(3)=val
c
      call vecnorm(rad,rad_norm,rmag)    ! normalized radius vector in inertial coordinates
c
c *** Gravity Gradient Torques ***
c
      call matvec(amat,rad_norm,rad_norm_b)      ! normalized radius vector in body coordinates
      call matvec(xmoi,rad_norm_b,xir)           ! [ I]{ r} , unit vector, body coordinates
      call vcross(rad_norm_b,xir,rxxir)          ! { r} X{ I}{ r}
c
      if(rmag.lt.1.0e-6) ggmul=0.0
      if(rmag.ge.1.0e-6) ggmul=3.0D0*gme/rmag**3

c
      t_gg(1)=ggmul*rxxir(1)                     ! gravity gradient torque in body coordinates
      t_gg(2)=ggmul*rxxir(2)
      t_gg(3)=ggmul*rxxir(3)
c
c *** Aerodynamic torque calculation at time=t1+tau
c
      do 2 i=1,ndp          !extracts scaler components of the input vector signal from the
      func_x(i)=uu2(4,i)    !common block "subsys2". The variable uu2(4-6,j) contains the aerodynamic
      func_y(i)=uu2(5,i)    !torque in body (c.m. origin) coordinates.
      func_z(i)=uu2(6,i)
    2 continue
c
      t1=0.0D0 ! time (tau) is measured from the left end point of the convergence interval
c
      call interp(ndp,t1,delt,tau,func_x,val)  !interpolates x-component of aerodynamic torque
      t_aero(1)=val
      call interp(ndp,t1,delt,tau,func_y,val)  !... y-component...
      t_aero(2)=val
      call interp(ndp,t1,delt,tau,func_z,val)  !... z-component...
      t_aero(3)=val

c *** Control System Torques ***
c
      t_control(1)=-(a0*phi(1)+a1*omega(1))    ! control torque in body coordinates (c.m. origin)
```

```
      t_control(2)=-(b0*phi(2)+b1*omega(2))
      t_control(3)=-(c0*phi(3)+c1*omega(3))
c
c *** Attitude dynamics
c
      call matvec(xmoi,omega,h)
      call vcross(omega,h,wxh)
      t_total(1)=t_GG(1)+t_aero(1)+t_control(1)-wxh(1) !disturbance plus gyroscopic torque in body frame
      t_total(2)=t_GG(2)+t_aero(2)+t_control(2)-wxh(2)
      t_total(3)=t_GG(3)+t_aero(3)+t_control(3)-wxh(3)
      call matvec(xmoi_inv,t_total,omegadot)           !angular acceleration of body frame in body
c                                                      !coordinates
c
c     Converting body rates to Euler angle derivatives
c
c     (Rotation sequence precludes 90deg angle about y-axis)
c
      sin_1=dsin(phi(1))
      cos_1=dcos(phi(1))
      sin_2=dsin(phi(2))
      cos_2=dcos(phi(2))
      tan_2=sin_2/cos_2
      cmat(1,1)=1.0D0
      cmat(1,2)=sin_1*tan_2
      cmat(1,3)=cos_1*tan_2
      cmat(2,1)=0.0D0
      cmat(2,2)=cos_1
      cmat(2,3)=-sin_1
      cmat(3,1)=0.0
      cmat(3,2)=sin_1/cos_2
      cmat(3,3)=cos_1/cos_2
      call matvec(cmat,omega,phidot)             ! phidot in non-orthogonal Euler-frame coordinates
c
c *** Changing back to mathematical notation from physical notation.
c

c
      xdot(1)=phidot(1)
```

```
      xdot(2)=phidot(2)
      xdot(3)=phidot(3)
      xdot(4)=omegadot(1)
      xdot(5)=omegadot(2)
      xdot(6)=omegadot(3)
c
      return
      end
```

```
xxxx
```

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      SUBROUTINE VCROSS(A,B,C)
C
C *** THIS ROUTINE COMPUTES THE CROSS PRODUCT OF TWO VECTORS
C     A AND B AND STORES THE RESULT AS C.  (C=AXB)
C
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C
      DIMENSION A(3),B(3),C(3)
C
      C(1)=A(2)*B(3)-B(2)*A(3)
      C(2)=B(1)*A(3)-A(1)*B(3)
      C(3)=A(1)*B(2)-B(1)*A(2)
      RETURN
      END

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      subroutine vdotpro(a,b,adotb)
c
c *** computes dot product of two vectors
c
      implicit double precision(a-h,o-z)
c
      dimension a(3),b(3)
c
      adotb=a(1)*b(1)+a(2)*b(2)+a(3)*b(3)
c
      return
```

```
      end
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx


      SUBROUTINE VECNORM(X,XNORM,XMAG)
C
C *** THIS ROUTINE NORMALIZES A VECTOR (3X1).
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C
      DIMENSION X(3),XNORM(3)
C
      XMAG=DSQRT(X(1)**2+X(2)**2+X(3)**2)
      if(xmag.le.0.0D0) go to 11
      DO 10 I=1,3
      XNORM(I)=X(I)/XMAG
   10 CONTINUE
C
      RETURN
c
   11 xmag=0.0D0
c
      do 12 i=1,3
      xnorm(i)=0.0D0
   12 continue
c
      return
      END
```

B.2 FORTRAN Listings, Including Main Program and All Subroutines Not Previously Listed in A.1, Recursive Feedback With Distributed Coupling (RF–D)

```
c
c     This is FORTRAN MAIN program "rfd.f",and is a variation of FORTRAN main program "rfc.f".
c     Development began 6/12/2000 by Doug Nixon, MSFC/TD55, in support of AEI. The main and subprogram
c     listings are contained in "aei/rfd"
c
c     This example of simulation by recursive feedback uses distributed coupling, and convergence is
c     checked on the minimum number of signals (aerodynamic forces and torques).
c
c     The purpose is to generate the dynamic solution of three coupled subsystems (orbit dynamics,
c     attitude dynamics, and aerodynamics) by recursive feedback, which involves distributed and
c     possibly multiple methods of numerical integration. The intent is to compare results generated
c     by a standard approach using a centralized single numerical integration method. The standard
c     approach has been implemented as FORTRAN MAIN program "rku.f", and the main and subprogram listings
c     are contained in "aei/rku".
c
c *** Declarations ************************************************************************
c
      implicit double precision(a-h,o-z)
c
      dimension uu1(6,100),yy1(6,100)                !input and output signals for three subsystems,
      dimension uu2(6,100),yy2(6,100)                !six signals per subsystem, 100 time points
      dimension uu3(6,100),yy3(6,100)
c
      dimension fb(6,100),fbp(6,100)                 !for convergence check
c
      dimension yy1i(6),yy2i(6),yy3i(6)              !initial conditions, 3 subsystems, 6 outputs
      dimension yy1f(6),yy2f(6),yy3f(6)              !final conditions, 3 subsystems, 6 outputs
c
      common/system/uu,yy,yyp     !system input/output/"previous" output, all vs. time
c
      common/subsys1/uu1,yy1      !orbit dynamics, i/o vs. time
      common/endcond1/yy1i,yy1f   !initial and final output conditions
c
      common/subsys2/uu2,yy2      !attitude dynamics, i/o vs. time
      common/endcond2/yy2i,yy2f   !initial and final output conditions
c
      common/subsys3/uu3,yy3      !aerodynamics, i/o vs. time
      common/endcond3/yy3i,yy3f   !initial and final output conditions
```

```
c
      common/simulation/ndp,delt,tol,res,t_start,      !all except some single-time-point utility routines
     1t_report,t_print,t_stop,nmt,nmtr,nit,
     2t_initial,t_final
c
      common/simulation2/del_report,del_print,ptol     !percent done report interval, print interval
c
      common/constants/rearth,gme,pi,cvdr              !recursive.f, orbit.f,sub1, attitude.f, sub2.f
c
c *** Definition of the fundamental variables ************************************************************
c
c ********************************************************************************************************
c
c     uu1(i,j)----- First subsystem input signal. Represents the ith component at the jth time
c                   point for the vector input signal to the first (orbit dynamics) susystem,
c                   and is also the first (top) partition of uu(i,j).
c
c     uu2(i,j)----- Second subsystem input signal. Represents the ith component at the jth time
c                   point for the vector input signal to the second (attitude dynamics) subsystem,
c                   and is also the second (middle) partition of uu(i,j).
c
c     uu3(i,j)----- Third subsystem input signal. Represents the ith component at the jth time
c                   point for the vector input signal to the third (aerodynamics) subsystem, and
c                   is also the third (bottom) partition of uu(i,j).
c
c ********************************************************************************************************
c
c     yy1(i,j)----- First subsystem output signal. Represents the ith component at the jth time
c                   point for the vector output signal from the first (orbit dynamics) subsystem,
c                   and is also the first (top) partition of yy(i,j).
c
c     yy2(i,j)----- Second subsystem output signal. Represents the ith component at the jth time
c                   point for the vector output signal from the second (attitude dynamics) subsystem,
c                   and is also the second (middle) partition of yy(i,j).
c
c     yy3(i,j)----- Third subsystem output signal. Represents the ith component at the jth time
c                   point for the vector output signal to the third (aerodynamics) subsystem, and is
c                   also the third (bottom)partition of yy(i,j).
```

```
c
c ***************************************************************************************************
c
c     yy1i(i) ----- Value of the output signal at the beginning of the current convergence interval
c                   for the FIRST (orbit dynamics) subsystem. Represents the ith component of the
c                   signal at the first time point, and is equivalent to yy1(i,1)
c
c     yy2i(i) ----- Value of the output signal at the beginning of the current convergence interval
c                   for the SECOND (attitude dynamics) subsystem. Represents the ith component of the
c                   signal at the first time point, and is equivalent to yy2(i,1)
c
c     yy3i(i) ----- Value of the output signal at the beginning of the current convergence interval
c                   for the THIRD (aerodynamics) subsystem. Represents the ith component of the signal
c                   at the first time point, and is equivalent to yy3(i,1)
c ***************************************************************************************************
c
c     yy1f(i) ----- Value of the output signal at the end of the current convergence interval for
c                   the FIRST (orbit dynamics) subsystem. Represents the ith component of the signal
c                   at the last time point, and is equivalent to yy1(i,ndp)
c
c     yy2f(i) ----- Value of the output signal at the end of the current convergence interval for
c                   the SECOND (attitude dynamics) subsystem. Represents the ith component of the
c                   signal at the last time point, and is equivalent to yy2(i,ndp)
c
c     yy3f(i) ----- Value of the output signal at the end of the current convergence interval for
c                   the THIRD (aerodynamics) subsystem. Represents the ith component of the vector
c                   signal yy3(i,j) at the last time point, and is equivalent to yy3(i,ndp)
c
c ******General Comments ***************************************************************************
c
c     The above nomenclature has been designed to allow implementation of recursive feedback via
c     FORTRAN in such a way that independence of the subsystem simulations can be supported. Definition
c     of submatrices (partitioning) facilitates independence of the subsystem simulations while providing
c     appropriate structure for implementation of the system-level algorithm. The fact that subsystem
c     data has been placed in COMMON with the MAIN (system-level) program does not necessarily mean
c     that all of that data is required or desired at the system level -- it may mean only that the
c     SUBROUTINE containing the subsystem simulation needs to have access to data that it generated
c     previously, and placing that data in COMMON is a way to provide that in FORTRAN.
```

```
c
c     Because of the distributed coupling configuration, information flows from one sub-simulation to the
c     next in a sequential manner. The order of execution is important, and parallel execution is not
c     possible.
c
c     This implementation assumes that all subsystem states are needed at the system level, and transfer of
c     final conditions to initial conditions after convergence is handled at that level. For this specific
c     system, the assumption is valid; however, in general,subsystem states can exist and remain at the
c     subsystem level without the necessity of knowing them at the system level. Because of this,
c     it would be more appropriate to define and "bookkeep" initial conditions at the subsystem level for a
c     more general implementation. In the more general implementation, transfer of final conditions from
c     one convergence interval to intial conditions of the next would be handled totally at the subsystem
c     level upon receipt of "command" from the system level.
c
c *** Establishing the output file ************************************************************************
c
      open(unit=1,file='rfd.out1',status='unknown')
c
c *** Defining system-level simulation parameters ********************************************************
c
      print*
      print*,'This is main program rfd.f. Solution'
      print*,'method is recursive feedback with'
      print*,'distributed coupling.'
      print*
      print*,'  Input the number of data (time) points'
      print*,'  per convergence interval, including'
      print*,'  both end (boundary) points:'
      print*
      read*, ndp
      print*
      print*,'  Input the convergence interval duration'
      print*,'  (sec.):'
      print*
      read*, dtmaj
      print*
c
      fac=1.0/float(ndp-1)      !ratio of integration time step to convergence interval duration
```

```
      delt=fac*dtmaj            !integration stepsize
c
      print*,'  The above data limits all integration'
      print*,'  stepsizes to ',delt,' sec.'
      print*,'  or less'
      print*
c
c *** Populating the subsystem databases ***************************************************************
c
      call orbit(1)       !defines parameter values (data) for subsystem 1 (orbit dynamics, mode 1)
c
      call attitude(1)    !defines parameter values (data) for subsystem 2 (orbit dynamics, mode 1)
c
      call aerodyn(1)     !defines parameter values (data) for subsystem 3 (orbit dynamics, mode 1)
c
      print*,'  Input the simulation stop time (sec.):'
      print*
      read*, t_stop
      print*
      print*,'  Input the data output(print) interval'
      print*,'  (sec.):'
      print*
      read*, del_print
      print*
      print*,'  Input percent done status interval'
      print*,'  (%):'
      print*
      read*, per_report
      print*
      del_report=(per_report/100.0)*t_stop
c
      nmtr=int(t_stop/dtmaj)+1   !number of major time (convergence) intervals required for completion of
c                               !the simulation. (Time may exceed t_stop by as much as one major interval)
c
      tol=(1.0D0)*1.0e-14         !relative
      res=(1.0D0)*1.0e-14         !absolute
      itmax=100                   !maximum allowable number of iterations
      ptol=1.0e-3                 !tolerance for print time
```

```
      print*,'  Run in progress.'
      print*
      call clock(n1)              !machine start time
c
c *** Initializing time and convergence interval count: ************************************************
c
      time=0.0      !time is measured from start of the simulation
      nmt=0         !number of major time intervals (convergence intervals) simulated
      t_print=0.0   !initializes print time
      t_report=0.0 !initializes percent done report time
c
c *** Setting up initial conditions for each sub-simulation at system simulation outset
c
      call orbit(2)      !mode 2, original initial conditions setup, orbit dynamics
c
      call attitude(2)   !mode 2, original initial conditions setup, attitude dynamics
c
c                        !no initial conditions setup for aerodynamics (algebraic system)
c
c ********************************************************************************************************
  100 continue !*****************COMPUTING INITIAL RECURSION STAGE DATA*************************************
c ********************************************************************************************************
c
      nit=0 !number of iterations initialized to zero for first (new) convergence interval
c
c *** Initializing the system-level feedback signal to zero: ******************************************
c
      do 1 i=1,6         !six scaler input signals per subsystem
      do 1 j=1,ndp       !ndp number of data (time) points per scaler signal
      uu1(i,j)=0.0D0     !initializing all subsystem input signals to zero (0)
      uu2(i,j)=0.0D0
      uu3(i,j)=0.0D0
    1 continue
c
c *** Running individual subsystem simulations over the convergence interval
c
      call orbit(3)       ! mode 3, run orbit dynamics simulation over convergence interval
c
```

```
      call coupler1
c
      call attitude(3)  ! mode 3, run attitude dynamics simulation over convergence interval
c
      call coupler2
c
      call aerodyn(3)   ! mode 3, run aerodynamics (algebraic) simulation over convergence interval
c
      call coupler3     !computes feedback signal from aerodynamics to orbit and attitude dynamics
c
      do 2 i=1,3        !sets initial recursion stage data to current values (aerodynamics, feedback)
      do 2 j=1,ndp
      fbp(i,j)=uu1(i,j)     !aerodynamic force, inertial coordinates
      fbp(i+3,j)=uu2(i+3,j) !aerodynamic torque, body coordinates
    2 continue
c
c ***************************************************************************************************
  200 continue !*********************RECURSION LOOP ENTRY POINT*************************************
c ***************************************************************************************************
      nit=nit+1            !updating the iterations count (number of recursion stages)
c
c *** Running individual subsystem simulations over the convergence interval
c
      call orbit(3)     ! mode 3, run orbit dynamics simulation over convergence interval
c
      call coupler1
c
      call attitude(3)  ! mode 3, run attitude dynamics simulation over convergence interval
c
      call coupler2
c
      call aerodyn(3)   ! mode 3, run aerodynamics (algebraic) simulation over convergence interval
c
      call coupler3
c
c *** Check for convergence ************************************************************************
c
      do 3 i=1,3
```

```
      do 3 j=1,3
      fb(i,j)=uu1(i,j)                          !aerodynamic force, inertial coordinates
      fb(i+3,j)=uu2(i+3,j)                      !aerodynamic torque, body coordinates
    3 continue
c
      call fastverge(6,fb,fbp,iconflag)         ! aerodynamic system convergence check (feedback)
c                                               ! iconflag=0 if converged, 1 if
c                                               ! notconverged
      idirect=iconflag+1
  500 format(/,2i5,/)
      go to (6,5) idirect          !6 for converged, 5 for not converged
c
c ***************************************************************************************************
    5 continue ! ***************** Not CONVERGED, ADVANCE RECURSION STAGE ********************************
c ***************************************************************************************************
c
      do 7 i=1,6            !number of scaler signals at recursive transfer point
      do 7 j=1,ndp          !number of time points in each convergence interval,
c                           !includes both end points
      fbp(i,j)=fb(i,j)      !sets previous recursion stage data to current values
    7 continue
c
      if(nit.ge.itmax) go to 300     !protection against persistent non convergence
c
      go to 200             !returns to "top" of the recursion loop to run subsystem simulations again
c
c ***************************************************************************************************
    6 continue ! ** CONVERGED, ADVANCE TO NEXT CONVERGENCE INTERVAL (or stop if simulation is complete) **
c ***************************************************************************************************
c
      call rfd_out            !appends system-level output signals to output file (rfd.out1)
      nmt=nmt+1               !increments number of major time (convergence) intervals completed
      time=nmt*dtmaj          !simulated problem time, for reference only
c
      if(nmt.eq.nmtr) go to 300   !end of run, time is greater than or equal to the stop time (t_stop)
c
c ***************************************************************************************************
c
```

```
c *** Proceed ("hand-off") to next major time (convergence) interval
c
c     In a more general setting, this would be done at the subsystem level (mode 4) so that the system
c     level is not required to have knowledge of all subsystem states.
c
      do 8 i=1,6              !extracts subsystem final conditions for the current convergence interval
      yy1f(i)=yy1(i,ndp)      !from the final time point of the system response associated with the current
      yy2f(i)=yy2(i,ndp)      !convergence interval. SEE PARAGRAPH NUMBER 3 of "General Comments" ABOVE.
      yy3f(i)=yy3(i,ndp)
    8 continue
c
      do 9 i=1,6
      yy1i(i)=yy1f(i)         !sets initial conditions (for all states) for the next convergence interval
      yy2i(i)=yy2f(i)         !equal to conditions reached at the end (final conditions) of current
      yy3i(i)=yy3f(i)         !convergence interval.
    9 continue
c
      go to 100               !run subsystem simulations for the first time across the new (next)
c                             !convergence interval.
  300 continue
c
      print*
      if(nit.ge.itmax) print*,'Max. allowable number of '
      if(nit.ge.itmax) print*,'iterations reached.'
c
      call clock(n2) !machine stop time
      irun=n2-n1     !run-time

      if(nmt.eq.nmtr) print*,'  Normal exit, run-time = ',irun,' sec.'
      print*
c
      close(unit=1,status='keep')
c
      stop
c
c ***************************************************************************************************
      end !********************************* END OF MAIN PROGRAM ***************************************
c ***************************************************************************************************
```

```
xxxx
```

```
      return
      end

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx


      subroutine coupler2
c
      implicit double precision(a-h,o-z)
c
      dimension uu1(6,100),yy1(6,100)
      dimension uu2(6,100),yy2(6,100)
      dimension uu3(6,100),yy3(6,100)
c
      dimension rad(3),vel(3)           !orbit radius and velocity vectors, inertial coordinates
      dimension rad_b(3),vel_b(3)       !orbit radius and velocity vectors, body coordinates
      dimension phi(3)                  !Euler angles, inertial frame to body frame, 3-2-1 sequence
      dimension amat(3,3)               !direction cosine matrix (d.c.m.), inertial to body
      dimension amat_t(3,3)             !transpose of amat(i,j)
c
      common/simulation/ndp,delt,tol,res,t_start,     !all except some single-time-point utility routines
     1t_report,t_print,t_stop,nmt,nmtr,nit,
     2t_initial,t_final
c
      common/subsys1/uu1,yy1
      common/subsys2/uu2,yy2
      common/subsys3/uu3,yy3
c
c *****************************************************************************************************************************
c ******************** Coupling from orbit and attitude dynamics to aerodynamics  *****************************
c *****************************************************************************************************************************
c
      do 10 j=1,ndp
```

```
c
      rad(1)=yy1(1,j)            !extracting orbit radius from subsystem 1 output
      rad(2)=yy1(2,j)
      rad(3)=yy1(3,j)
c
      phi(1)=yy2(1,j)            !extracting attitude (Euler) angles from subsystem 2 output signal
      phi(2)=yy2(2,j)
      phi(3)=yy2(3,j)
c
      call dcm321(phi,amat)     !computes d.c.m that transforms vectors from inertial to body coordinates
      call tran3(amat,amat_t)   !computes d.c.m that transforms vectors from body to inertial coordinates
c
c *** from orbit dynamics to aerodynamics
c
      call matvec(amat,rad,rad_b) !transforms orbit radius vector to body coordinates
c
      uu3(1,j)=rad_b(1)            !feedback signal is orbit radius vector in body coordinates
      uu3(2,j)=rad_b(2)
      uu3(3,j)=rad_b(3)
c
c
      vel(1)=yy1(4,j)              !variable vel(i) is used for listing readability only
      vel(2)=yy1(5,j)
      vel(3)=yy1(6,j)
c
      call matvec(amat,vel,vel_b) !transforms orbit velocity vector to body coordinates

      uu3(4,j)=vel_b(1)            !feedback signal is orbit velocity vector in body coordinates
      uu3(5,j)=vel_b(2)
      uu3(6,j)=vel_b(3)
c
   10 continue
c
      return
      end

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      subroutine coupler3
c
      implicit double precision(a-h,o-z)
c
      dimension uu1(6,100),yy1(6,100)
      dimension uu2(6,100),yy2(6,100)
      dimension uu3(6,100),yy3(6,100)
c
      dimension f_aero(3)                  !aerodynamic force (drag), inertial coordinates
      dimension f_aero_b(3), t_aero_b(3)  !aerodynamic force and torque, body frame
      dimension f_aero_g(3), t_aero_g(3)  !aerodynamic force and torque, geometric frame
      dimension rxf(3)                     !aerodynamic force transfer couple (from geometric origin to c.m.)
      dimension phi(3)                     !Euler angles, inertial frame to body frame, 3-2-1 sequence
      dimension amat(3,3)                  !direction cosine matrix (d.c.m.), inertial to body
      dimension amat_t(3,3)                !transpose of amat(i,j)
      dimension xmoi(3,3),xmoi_inv(3,3)    !inertia matrix, inverse inertia matrix
      dimension rcm(3)                     !center of mass location, geometric coordinates
c
      common/simulation/ndp,delt,tol,res,t_start,     !all except some single-time-point utility routines
     1t_report,t_print,t_stop,nmt,nmtr,nit,
     2t_initial,t_final
c
      common/mass/xmass,xmoi,xmoi_inv,rcm             !orbit.f, sub1.f, attitude.f, sub2.f, coupler.f
c
      common/subsys1/uu1,yy1
      common/subsys2/uu2,yy2
      common/subsys3/uu3,yy3
c
c ****************************************************************************************************************************
c ************************ Coupling from aerodynamics to orbit and attitude dynamics *************************
c ****************************************************************************************************************************
c
```

```
      do 10 j=1,ndp
c
c *** from aerodynamics to orbit dynamics:
c
      f_aero_b(1)=yy3(1,j)      !extracting aerodynamic (drag) force from system output signal
      f_aero_b(2)=yy3(2,j)      !computed to act at origin of geometric body frame,
      f_aero_b(3)=yy3(3,j)      !but can be assummed to act at c.m. for this purpose
c
      phi(1)=yy2(1,j)            !extracting attitude (Euler) angles from the system output signal
      phi(2)=yy2(2,j)
      phi(3)=yy2(3,j)
c
      call dcm321(phi,amat)     !computes d.c.m that transforms vectors from inertial to body coordinates
      call tran3(amat,amat_t)   !computes d.c.m that transforms vectors from body to inertial coordinates
c
      call matvec(amat_t,f_aero_b,f_aero) !transforms force vector from body to inertial coordinates
c
      uu1(1,j)=f_aero(1)     !feedback signal is aerodynamic force (drag) in inertial coordinates,
      uu1(2,j)=f_aero(2)     !and is assummed to act at the satellite center of mass (c.m.)
      uu1(3,j)=f_aero(3)
c
      uu1(4,j)=0.0D0         !slot holders, nothing yet identified
      uu1(5,j)=0.0D0
      uu1(6,j)=0.0D0
c
c *** from aerodynamics to attitude dynamics:
c
      f_aero_g(1)=yy3(1,j)    !aerodynamic (drag) force; acts at origin of geometric body frame.
      f_aero_g(2)=yy3(2,j)    !f_aero_g(i) and f_aero_b(i) are numerically indentical, but their
      f_aero_g(3)=yy3(3,j)    !physical interpretations (points of action) may be different
c
      t_aero_g(1)=yy3(4,j)    !aerodynamic torque with force at origin of geometric body frame
      t_aero_g(2)=yy3(5,j)
      t_aero_g(3)=yy3(6,j)
c
      call vcross(rcm,f_aero_g,rxf)           ! torque of drag force about c.m.
c
      t_aero_b(1)=t_aero_g(1)+rxf(1)          ! transferring drag force to c.m. and adding an
```

```
      t_aero_b(2)=t_aero_g(2)+rxf(2)          ! eqivalent couple.
      t_aero_b(3)=t_aero_g(3)+rxf(3)
c
      uu2(4,j)=t_aero_b(1)
      uu2(5,j)=t_aero_b(2)
      uu2(6,j)=t_aero_b(3)
c
   10 continue
c
      return
      end

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx


      subroutine fastverge(ndim,yy3,yy3p,iconflag) !system-level convergence check
c
      implicit double precision(a-h,o-z)
c
      dimension yy3(6,100),yy3p(6,100)
      dimension ynew(100),yold(100)
c
      common/simulation/ndp,delt,tol,res,t_start,t_report,t_print,
     1t_stop,nmt,nmtr,nit,t_initial,t_final
c
c **********************************************************************************************************************
c ********************************************** begin check loop ****************************************************
c **********************************************************************************************************************
c
      iconflag=0            !initializes system-level flag to "converged"

      do 20 i=1,ndim        !loads vector elements for scalar signal convergence check
      do 19 j=1,ndp
```

```
c
      ynew(j)=yy3(i,j)
      yold(j)=yy3p(i,j)
c
   19 continue
c
      call convrg(ynew,yold,ndp,tol,res,iflag) ! iflag=1 for non-convergence.
c
      iconflag=iconflag+iflag !computes status of system-level convergence
c
   20 continue
c
      if(iconflag.gt.0) iconflag=1 !routine returns 0 or 1 for iconflag
c
c ******************************************************************************************************
c ***************************************** end check loop *********************************************
c ******************************************************************************************************
c
      return
c
      end


xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx


      subroutine rfd_out
c
      implicit double precision(a-h,o-z)
c
```

```
      dimension uu1(6,100),yy1(6,100)
      dimension uu2(6,100),yy2(6,100)
      dimension uu3(6,100),yy3(6,100)
c
      dimension zz(18,100)        !single output array at system level
c
      dimension rad(3),rad_n(3)   !for processing orbit data to obtain altitude
c
      dimension rcm(3), f_aero(3), t_aero(3)      !added 9/21/2000, for aero torque output
      dimension xmoi(3,3),xmoi_inv(3,3),rcm(3)    !moments of inertia, about c.m, c.m. location vector
c
      dimension phi(3) !Euler angles, added 10/2/00. needed here to limit values for output
c
      common/subsys1/uu1,yy1
      common/subsys2/uu2,yy2
      common/subsys3/uu3,yy3
c
      common/simulation/ndp,delt,tol,res,t_start,  !all except some single-time-point utility routines
     1t_report,t_print,t_stop,nmt,nmtr,nit,
     2t_initial,t_final
c
      common/simulation2/del_report,del_print,ptol !percent done report interval, print interval
c
      common/constants/rearth,gme,pi,cvdr          !recursive.f, orbit.f,sub1, attitude.f, sub2.f
c
      common/mass/xmass,xmoi,xmoi_inv,rcm          !orbit.f, sub1.f, attitude.f, sub2.f, coupler.f
C                                                  !added 9/21/2000, for aero torque output
      do 3 i=1,6
      do 3 j=1,ndp
c
      zz(i,j)=yy1(i,j)
      zz(i+6,j)=yy2(i,j)
      zz(i+12,j)=yy3(i,j)
c
    3 continue
c
      do 4 i=1,3
      do 4 j=1,ndp
```

```
      phi(i)=zz(i+6,j)                ! unlimited angle (10/2/00)
      call dcm321(phi,amat)
      call dcm2ang321(amat,phi)
      zz(i+6,j)=phi(i)/cvdr           ! limited angle   (10/2/00)
      zz(i+9,j)=yy2(i+3,j)/cvdr
c
    4 continue
c
      do 5 j=1,ndp
      f_aero(1)=zz(13,j)                !added 9/21/2000 for aero. torque computaion for output
      f_aero(2)=zz(14,j)
      f_aero(3)=zz(15,j)
      call vcross(rcm,f_aero,t_aero)
      zz(16,j)=t_aero(1)
      zz(17,j)=t_aero(2)
      zz(18,j)=t_aero(3)
    5 continue
c
      time1=nmt*(ndp-1)*delt    !nmt=number of major time (convergence) intervals complete
c                               !ndp=number of data points per major interval, including both end points
c                               !delt=time between data points (also integration stepsize)
      time=time1                !initializing time to first point of convergence interval
c
      do 1000 i=1,ndp-1                       ! begin first write loop
c
        do 1 k=1,3                                  !post processing orbit data for altitude
        rad(k)=yy1(k,i)
    1   continue
        call vecnorm(rad,rad_n,rmag)
        alt=rmag/1000.0D0-rearth  !altitude, kilometers
c
        if(time.ge.t_print) write(1,100) time,(zz(j,i),j=1,18),alt,nit
        if(time.ge.t_print) t_print=t_print+del_print
c
        if(time.ge.t_report) ipd=int((time/t_stop)*100.0)
        if(time.ge.t_report) print*,'Percent complete = ',
     1  ipd
        if(time.ge.t_report) t_report=t_report+del_report
```

```
  100 format(20E20.10,i5)
      time=time+delt
 1000 continue
c
      return
      end
```

B.3 FORTRAN Listings, Including Main Program and All Subroutines Not Previously Listed in A.1, Unified Approach, Runge-Kutta Method (RK–U)

```
c
c *** This is main program "rku.f". Construction began 5/9/2000 by Doug Nixon, TD55, MSFC.
c     It is designed to produce a standard-method benchmark to support an evaluation of
c     recursive feedback as a multi-discipline time-domain simulation solution method.
c     The test-case is a system involving orbital mechanics, vehicle rotational dynamics
c     and aerodynamics."rku.f" is the main driver for the twelve-state dynamic system which
c     is integrated by "rksys.f", a fourth-order Runge-Kutta subroutine. "sub.f" defines the
c     dynamics in first-order state-variable form. Listings are contained in "aei/rku".
c
      implicit double precision (a-h,o-z)
c
      dimension xmoi(3,3),xmoi_inv(3,3),rcm(3)
      dimension r(3),v(3),phideg(3),phi(3),omegadeg(3),omega(3)
      dimension x(12),xdot(12)
c
      dimension cpfront(3),cpright(3),cpbottom(3)      !constant aerodynamic system parameters
      dimension cprear(3),cpleft(3),cptop(3)
      dimension afront(3),aright(3),abottom(3)
      dimension arear(3),aleft(3),atop(3)
c
      common/constants/gme
      common/thetasave/theta1,add
      common/convert/cvdr
      common/mass/xmass,xmoi,xmoi_inv,rcm
      common/gains/a0,a1,b0,b1,c0,c1
      common/altitude/rmag,rearth
c
      common/aeroconst/ cpfront,cpright,cpbottom,cprear,cpleft,    !in common with aerocomp
     1cptop,afront,aright,abottom,arear,aleft,atop,cd,rho
c
      print*
      print*,'       This main program rku.f. It uses a'
      print*,'       conventional centralized integration'
      print*,'       method.(Runge-Kutta).'
      print*
      print*,'       Input the integration stepsize,'
      print*,'       (sec.):'
      read*, delt
```

```
      print*,'       Input the output (print) interval,'
      print*,'       (sec):'
      read*, delprint
      print*,'       Input the stop time, (sec)'
      read*, tstop
      print*,'       Run in progress.'
      call clock(n1)                    ! machine start time
c
c *** Numerically Defining Constants: ***
c
      gravconst=(6.673D0)*(1.0e-11)     ! Constant of Gravitation,
                                        ! m**3/kg-sec**2, CRC, page 6
c
      rearth= 6367.65D0                 ! Earth's radius,(polar+equatorial)/2.0,
                                        ! km, CRC, page 6
c
      dearth=5522.0D0                   ! Average density of the earth,
                                        ! kg/m**3, CRC, page 6
      pi=datan2(0.D0,-1.D0)             ! Computes pi to machine precision
      cvdr=pi/180.0D0                   ! Converts degrees to radians
c
      volume=(4.0D0/3.0D0)*pi*(rearth*1000.0D0)**3  ! Volume of earth, m**3
c
      xmearth=dearth*volume             ! Mass of the earth, kg
c
      gme=gravconst*xmearth             ! Gravity Constant times mass of the earth.
c
c *** Numerically Defining System Data: ***
c
      alt=300.0D0                       ! altitude, kilometers

      xmass=10000.0D0                   ! spacecraft mass, kilograms

      xmoi(1,1)=11250.0D0               ! spacecraft moments of inertia,
      xmoi(1,2)=0.0D0                   ! kilogram-meters**2
      xmoi(1,3)=0.0D0
      xmoi(2,1)=0.0D0
      xmoi(2,2)=193125.0D0              ! 3m diameter x 15m length uniform-
```

```
      xmoi(2,3)=0.0D0                  ! density right-circular cylinder
      xmoi(3,1)=0.0D0
      xmoi(3,2)=0.0D0
      xmoi(3,3)=193125.0D0*.9 !IZZ=.9*Iyy
      call inv3(xmoi,xmoi_inv)
c
      rcm(1)=1.0D0                     ! center of mass location in geometric body
      rcm(2)=0.1                     ! coordinates, meters !offset in y and z 7/13/00
      rcm(3)=0.1
c
c      control=1.0D0  !control on
      control=0.0D0  !control off
      a0=1000.0D0*control          ! control gain, x-axis position, n-m/rad
      a1=100.0D0*control           ! control gain, x-axis rate, n-m-sec/rad
      b0=1000.0D0*control          ! y-axis
      b1=100.0D0*control
      c0=1000.0D0*control          ! z-axis
      c1=100.0D0*control
c
c *** Setting up fixed aerodynamic data, in common/aeroconst/ with rksys/sub/aerocomp
c
      xl=15.0D0              ! length, meters (box shape, x-axis dimension)
      w=4.0D0                ! width, y-axis dimension
      h=3.0D0                ! height, z-axis dimension
      ax=w*h                 ! end area, normal to x-axis (+ "front")
      ay=xl*h                ! side area, normal to y-axis (+ "right side")
      az=xl*w                ! top or bottom area, normal to (+ z-axis "bottom")
c
c     Work with six facets ----
c
      do 9 i=1,3         ! setting all vector components to zero
c
      afront(i)=0.0D0   ! area vectors, normal to facet surfaces, through
      arear(i)=0.0D0    ! centers, positive (point) outward from center of
      aright(i)=0.0D0   ! box, back sides not considered (no contribution)
      aleft(i)=0.0D0
      atop(i)=0.0D0
      abottom(i)=0.0D0
```

```
c
      cpfront(i)=0.0D0 ! centers of pressure location vectors, from geometric
      cprear(i)=0.0D0  ! center of box to center of each facet
      cpright(i)=0.0D0
      cpleft(i)=0.0D0
      cptop(i)=0.0D0
      cpbottom(i)=0.0D0
c
    9 continue
c
      afront(1)=ax   ! setting non-zero components of facet area vectors
      arear(1)=-ax
      aright(2)=ay
      aleft(2)=-ay
      abottom(3)=az
      atop(3)=-az
c
      cpfront(1)=xl/2.0D0    !setting non-zero components of facet
      cprear(1)=-xl/2.0D0    !centers-of-pressure location vectors
      cpright(2)=w/2.0D0
      cpleft(2)=-w/2.0D0
      cptop(3)=-h/2.0D0
      cpbottom(3)=h/2.0D0
c
      cd=2.25                ! drag coefficient, same for all facets,
c                            ! not a function of direction
c
      rho=1.95E-11 ! kg/m**3, Wertz,inside back cover, 300km altitude , mean value
c
c
c *** Setting Initial Conditions, assuming a circular orbit at a given altitude: ***
c
      rmag=(rearth+alt)*1000.0D0          ! Radius magnitude, meters
c
      r(1)=rmag*dcos(30.0d0*cvdr)         ! Assumes a 30 deg "beta angle", (between ecliptic
      r(2)=0.0d0                          ! and orbit planes)
      r(3)=rmag*dsin(30.0d0*cvdr)
```

```
c
      vmag=dsqrt(gme/rmag)                ! Velocity magnitude for a circular orbit, meters/sec
c
      rcube=rmag**3
      wsq=gme/rcube
      cperiod=2.0d0*pi/dsqrt(wsq)         ! Period for circular orbit
c
      v(1)=0.0                            ! Velocity vector, circular orbit, meters/sec
      v(2)=vmag
      v(3)=0.0
c
      phideg(1)=0.0                       ! Zero initial attitude error, degrees.
      phideg(2)=0.0
      phideg(3)=0.0
c
      omegadeg(1)=0.0                     ! Zero initial body rates, degrees/sec
      omegadeg(2)=0.0
      omegadeg(3)=0.0
c
c *** Changing Angular variables from degrees to radians: ***
c
      phi(1)=phideg(1)*cvdr                ! deg to rad (attitude error)
      phi(2)=phideg(2)*cvdr
      phi(3)=phideg(3)*cvdr
      omega(1)=omegadeg(1)*cvdr            ! deg/sec to rad/sec (body rates)
      omega(2)=omegadeg(2)*cvdr
      omega(3)=omegadeg(3)*cvdr
c
c *** Setting up the initial state vector (12 states): ***
c
      do 10 i=1,3
      x(i)=r(i)             ! states 1-3,    orbit radius, meters
      x(i+3)=v(i)           ! states 4-6,    orbit velocity, meters/sec
      x(i+6)=phi(i)         ! states 7-9,    body angles (attitude errors), radians
      x(i+9)=omega(i)       ! states 10-12,  body rates, radians/sec
   10 continue
c
c *** Initializing time, opening data files, etc.
```

```
c
      num=12                ! number of system states
      time=0.0D0            ! initial time (problem start time), sec
      call sub(xdot,x,time)! computes dependent variables for initial output
      tprint=time           ! initializing time to print output data set
      ptol=.001*delt        ! tolerance on time for print
c
      theta1=datan2(r(2),r(1))! initial value of true anomaly, radians
      theta1=theta1/cvdr      ! convert to degrees
      add=0.0D0 !             ! initial value for multiple revolution correction
c
      if(theta1.lt.0.0D0) theta1=theta1+360.0D0 ! full cirle before reset
c
      open(unit=1,file='rku.out1',status='unknown') !opens data output file
c
   20 continue !***************** begin integration loop *************************
c
      if(abs(time-tprint).lt.ptol) call rku_out(time,x,delprint,tprint) !writes to file, updates tprint
c
      call rksys(time,x,num,delt) ! propagates states, increments time
c
      if(time.lt.tstop) go to 20  ! return to top of integration loop, or stop
c
      continue !****************** end integration loop *************************
c
      call rku_out(time,x,delprint,tprint)                    !writes final data set to file
c
      close(unit=1,status='keep') !closes data output file
c
      call clock(n2)               !machine stop time
      iruntime=n2-n1
      print*
      print*,'       Normal exit, run-time = ',iruntime,' sec.'
c
      stop
      end
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

      subroutine aerocomp(vel,amat,force,torque)
c
c     Developed starting 5/26/2000 by Doug Nixon, TD55, MSFC to compute
c     aerodynamic forces and torques for the "truth model" for recursive
c     feedback simulation method comparison for AEI study.
c
c     rho       = Atmospheric density, kg/m**3
c     vel(i)    = Velocity vector in inertial coordinates, meters/sec
c     amat(i,j) = Direction cosine matrix, transforms vectors from inertial to
c                 body coordinates
c     force(i)  = Aerodynamic force, body coordinates, newtons
c     torque(i) = Aerodynamic torque, body coordinates, newton-meters
c
      implicit double precision (a-h,o-z)
c
      dimension vel(3),amat(3,3),force(3),torque(3)
      dimension vnorm(3),vnorm_b(3)
c
      dimension cpfront(3),cpright(3),cpbottom(3)      !constant system parameters computed or defined
      dimension cprear(3),cpleft(3),cptop(3)           !in the main program, standard.f
      dimension afront(3),aright(3),abottom(3)
      dimension arear(3),aleft(3),atop(3)
c
      common/aeroconst/ cpfront,cpright,cpbottom,cprear,cpleft,    !in common with standard
     1cptop,afront,aright,abottom,arear,aleft,atop,cd,rho

      dimension aproj(6),drag(6),facefor(6,3),facetor(6,3)  !internally computed variables
      dimension tforce(3),ttorq(3)                          !output variables
c
c *** Aerodynamic forces ***
c
      call vecnorm(vel,vnorm,vmag)     ! normalize velocity vector in inertial
```

```
c                                     ! coordinates
c
      call matvec(amat,vnorm,vnorm_b)  ! normalized velocity vector in body
c                                     ! coordinates
c
c *** computing magnitude drag for each of six facets
c
      call vdotpro(afront,vnorm_b,aproj(1))     ! computing dot product of each area
      call vdotpro(aright,vnorm_b,aproj(2))     ! vector with normalized velocity
      call vdotpro(abottom,vnorm_b,aproj(3))    ! (projected areas)
      call vdotpro(arear,vnorm_b,aproj(4))
      call vdotpro(aleft,vnorm_b,aproj(5))      ! these statements define the facet
      call vdotpro(atop,vnorm_b,aproj(6))       ! numbering system
c
      do 20 i=1,6
      if(aproj(i).le.0.0D0) aproj(i)=0.0D0   ! negative projection implies ith facet
   20 continue                               ! not exposed to velocity vector
c
      fac=(0.5D0)*rho*(vmag**2)*cd           ! scaling by dynamic pressure times
      do 21 i=1,6                            ! drag coefficient
      drag(i)=fac*aproj(i)
   21 continue
c
      do 22 i=1,6
      do 22 j=1,3
      facefor(i,j)=-drag(i)*vnorm_b(j)       ! this is jth component of force on the ith
   22 continue                               ! facet, expressed in body coordinates
c
      do 23 i=1,3
      force(i)=0.0D0
      do 23 j=1,6
      force(i)=force(i)+facefor(j,i)  ! total aerodynamic force, body coordinates, newtons
   23 continue
c
c *** Aerodynamic Torques ***************************************************
c
      do 30 i=1,3                          ! front facet contribution
      tforce(i)=facefor(1,i)
```

```
   30 continue
      call vcross(cpfront,tforce,ttorq)
      do 31 i=1,3
      facetor(1,i)=ttorq(i)
      torque(i)=torque(i)+ttorq(i)
   31 continue
c
      do 32 i=1,3                          ! adding right facet contribution
      tforce(i)=facefor(2,i)
   32 continue
      call vcross(cpright,tforce,ttorq)
      do 33 i=1,3
      facetor(2,i)=ttorq(i)
   33 continue
c
      do 34 i=1,3                          ! adding bottom facet contribution
      tforce(i)=facefor(3,i)
   34 continue
      call vcross(cpbottom,tforce,ttorq)
      do 35 i=1,3
      facetor(3,i)=ttorq(i)
   35 continue
c
      do 36 i=1,3                          ! adding rear facet contribution
      tforce(i)=facefor(4,i)
   36 continue
      call vcross(cprear,tforce,ttorq)
      do 37 i=1,3
      facetor(4,i)=ttorq(i)
   37 continue
c
      do 38 i=1,3                          ! adding left facet contribution
      tforce(i)=facefor(5,i)
   38 continue
      call vcross(cpleft,tforce,ttorq)
      do 39 i=1,3
      facetor(5,i)=ttorq(i)
   39 continue
```

```
c
      do 40 i=1,3                           ! adding top facet contribution
      tforce(i)=facefor(6,i)
   40 continue
      call vcross(cptop,tforce,ttorq)
      do 41 i=1,3
      facetor(6,i)=ttorq(i)
   41 continue
c
      do 42 i=1,3
      torque(i)=0.0D0
      do 42 j=1,6
      torque(i)=torque(i)+facetor(j,i)  ! total aerodynamic torque, body
   42 continue                          ! coordinates, newton-meters
c
      return
      end

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx


      SUBROUTINE RKSYS(X,Y,N,H)
C
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C
C     THIS SUBROUTINE PERFORMS ONE STEP IN THE RUNGE-KUTTA
C     INTEGRATION OF A SYSTEM OF FIRST ORDER DIFFERENTIAL
C     EQUATIONS OF THE FORM Y'(X)=FD(X,Y), WHERE Y AND FD
C     ARE VECTOR FUNCTIONS, AND X IS THE INDEPENDENT
C     VARIABLE.  THE ALGORITHM IS DESCRIBED BY THE FOLLOWING
C     VECTOR RELATIONS:
C
C          F1=FD(X,Y),              YT=Y+F1*H/6
```

```
C           F2=FD(X+H/2,Y+F1*H/2),   YT=YT+F2*H/3
C           F3=FD(X+H/2,Y+F2*H/2),   YT=YT+F3*H/3
C           F4=FD(X+H,Y+F3*H),        Y=YT+F4*H/6
C            X=X+H,                  NEXT STEP
C
      DIMENSION Y(12),YSTART(12),YTEMP(12),FDERIV(12)
      H2=H/2.0
      H3=H/3.0
      H6=H/6.0
      X0=X
      DO 200 I=1,N
  200 YSTART(I)=Y(I)
      CALL SUB(FDERIV,Y,X)
      DO 300 I=1,N
      YTEMP(I)=YSTART(I)+FDERIV(I)*H6
  300 Y(I)=YSTART(I)+FDERIV(I)*H2
C
C *** END OF FIRST PASS ***
C
      X=X0+H2
      CALL SUB(FDERIV,Y,X)
      DO 400 I=1,N
      YTEMP(I)=YTEMP(I)+FDERIV(I)*H3
  400 Y(I)=YSTART(I)+FDERIV(I)*H2
C
C *** END OF SECOND PASS ***
C
      CALL SUB(FDERIV,Y,X)
      DO 500 I=1,N
      YTEMP(I)=YTEMP(I)+FDERIV(I)*H3
  500 Y(I)=YSTART(I)+FDERIV(I)*H
C
C *** END OF THIRD PASS ***
C
      X=X0+H
      CALL SUB(FDERIV,Y,X)
      DO 600 I=1,N
  600 Y(I)=YTEMP(I)+FDERIV(I)*H6
```

```
C
C *** END OF FOURTH PASS ***
C
C     AT THIS POINT ONE INTEGRATION STEP HAS BEEN COMPLETED.
C     Y(X) HAS BEEN REPLACED BY Y(X+H), AND
C     X HAS BEEN REPLACED BY X+H.
C
      RETURN
      END

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx


      subroutine rku_out(time,x,delprint,tprint)
c
c *** writes state data for main program "rku.f"
c
      implicit double precision (a-h,o-z)
c
      dimension x(12),xcv(12),rad(3),rad_norm(3),vel(3),vel_norm(3)
      dimension f_aero(3),f_aero_b(3),f_aero_n(3)
      dimension t_aero_b(3)
      dimension f_grav(3)
      dimension t_gg(3),t_cont(3)
      dimension facefor(6,3),facetor(6,3)
      dimension phi(3),amat(3,3)
c
      common/constants/gme
      common/thetasave/theta1,add
      common/convert/cvdr
      common/altitude/rmag,rearth
c
      common/aeroforce/f_aero,f_aero_b    ! Added a fifth (ultimately redundant) call to sub in rksys.f
```

```
      common/aerotorq/t_aero_b            ! so that this data is current at print time. (6/27/00)
c
c
c ***************************************************************************************************
c ******************************** Post processing for a single time point ********************************
c ************************** (supports standard centralized integration method) ************************
c
***************************************************************************************************
c
      fgmag=dsqrt(f_grav(1)**2+f_grav(2)**2+f_grav(3)**2) !gravity force magnitude, newtons
c
      famag=dsqrt(f_aero(1)**2+f_aero(2)**2+f_aero(3)**2) !aerodynamic force (drag)
c                                                         !magnitude, newtons
c
      alt=(rmag/1000.0D0-rearth)                          !altitude, kilometers
c
      theta=datan2(x(2),x(1))                             !true anomaly, radians
      theta=theta/cvdr                                    !converting to degrees
      if(theta.lt.0.0) theta=theta+360.0                  !range=0 to 360 deg.
c
      theta2=theta+add
      if(theta2.ge.theta1) go to 1
         add=add+360.0D0                 !updates range correction factor after each revolution
         theta2=theta2+360.0
c
    1 continue
c
c *** limiting range of Euler angles for output:  (added 9/20/2000)
c
      do 3 i=1,3
      phi(i)=x(i+6)
    3 continue
      call dcm321(phi,amat)
      call dcm2ang321(amat,phi)
      do 2 i=1,3              ! renaming the state variables, converting angular states to degree units
c
      xcv(i)=x(i)             ! radius (meters)
      xcv(i+3)=x(i+3)         ! velocity (meters/sec)
```

```
      xcv(i+6)=phi(i)/cvdr    ! limited Euler angles (deg)
      xcv(i+9)=x(i+9)/cvdr    ! body rates (deg/sec)
    2 continue
c
c
**********************************************************************************************************
c *************************************** End post processing
***********************************************
c
**********************************************************************************************************
c ****************************************** Write to file
**************************************************
c
**********************************************************************************************************
c
      write(1,100) time,(xcv(i),i=1,12),          !time, states, true anomaly (reseting & continuous)
     1(f_aero_b(i),i=1,3),(t_aero_b(i),i=1,3),
     2 alt,theta,theta1
c
  100 format(22e20.10)                            !file name is 'standard.out1'
c
      tprint=tprint+delprint            !compute time for output of next data set
c
      theta1=theta2                     !update continuous true anomaly for future reference
c
      return
      end

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx


      subroutine sub(xdot,x,time)
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
```

```
c
c *** Construction began 4/28/2000, by Doug Nixon, TD55. The purpose is to
c     supply the dynamics xdot=F(x,time) for integration by rksys, a
c     fourth-order Runge-Kutta routine. The problem is to set up a
c     standard-method simulation of a simple multidisciplined system involving
c     orbital dynamics, vehicle attitude dynamics, and aerodynamics.
c
      implicit double precision (a-h,o-z)
c
      common/constants/gme
      common/mass/xmass,xmoi,xmoi_inv,rcm
      common/gains/a0,a1,b0,b1,c0,c1
      common/tgg/t_gg
      common/tcontrl/t_cont
      common/gravforce/f_grav
      common/aeroforce/f_aero,f_aero_b
      common/aerotorq/t_aero_b
      common/altitude/rmag,rearth
c
c *** definition of variables in common:
c
c       gme  : universal gravitational constant times mass of the earth (lb-ft**2/slug)
c       xmass: spacecraft mass (slugs)
c       xmoi : spacecraft moment of inertia tensor (3x3 matrix, slug-ft**2)
c
      dimension xmoi(3,3),xmoi_inv(3,3),rcm(3)
      dimension x(12),xdot(12)
      dimension r(3),rnorm(3),rbody(3),rdot(3)
      dimension v(3),vnorm(3),vbody(3),vdot(3)
      dimension phi(3),phidot(3),omega(3),omegadot(3)
      dimension f_grav(3)
      dimension t_gg(3)
      dimension f_aero(3),f_aero_b(3)
      dimension t_aero_b(3)
      dimension t_cont(3)
      dimension t_tot(3)
      dimension cmat(3,3)
      dimension amat(3,3),amat_t(3,3)
```

```
      dimension temp1(3),temp2(3)
c
c     The system has six degrees of freedom, or twelve states. The System states are
c     defined as follows:
c
c     Orbit States (Vehicle Position States):
c
c       x(1)=r(1): x-component of orbit radius vector in inertial coordinates,
c                  from origin of inertial system to c.m. of spacecraft (meters)
c       x(2)=r(2): y-component
c       x(3)=r(3): z-component
c
c       x(4)=v(1): x-component of spacecraft velocity vector in inertial coordinates
c                 (meters/second)
c       x(5)=v(2): y-component
c       x(6)=v(3): z-component
c
c     Vehicle Rotational States:
c
c       x(7)=phi(1): Euler angle rotation from inertial to body system, about x-axis,
c                    3rd in sequence. (radians)
c       x(8)=phi(2): About y-axis, 2nd in sequence.
c       x(9)=phi(3): About z-axis, 1st in sequence.
c
c       x(10)=omega(1):  Body rate, x-axis (radians/second)
c       x(11)=omega(2):  Body rate, x-axis
c       x(12)=omega(3):  Body rate, x-axis
c
c     Aerodynamics States:
c
c       No states are defined because the aerodynamics model has been reduced to
c       an algebraic system.
c
c ***************************************************************************
c
      r(1)=x(1)          ! Changing from mathematical to physical notation.
      r(2)=x(2)
      r(3)=x(3)
```

```
      v(1)=x(4)
      v(2)=x(5)
      v(3)=x(6)
      phi(1)=x(7)
      phi(2)=x(8)
      phi(3)=x(9)
      omega(1)=x(10)
      omega(2)=x(11)
      omega(3)=x(12)
c
      call dcm321(phi,amat)     !computes transformation (DCM) from inertial
c                               !to body coordinates
c
      call tran3(amat,amat_t)   !transformation from body to inertial coordinates
c
c *** Gravity forces ***
c
      call vecnorm(r,rnorm,rmag)    ! normalized radius vector in inertial coordinates
      gmag=gme*xmass/rmag**2
c
      f_grav(1)=-gmag*rnorm(1)         ! result in inertial coordinates
      f_grav(2)=-gmag*rnorm(2)
      f_grav(3)=-gmag*rnorm(3)

c
c *** Gravity Gradient Torques ***
c
      ggmul=3.0D0*gme/rmag**3
      call matvec(amat,rnorm,rbody)    ! normalized radius vector in body coordinates
      call matvec(xmoi,rbody,temp1)         ! [ I]{ rbody}
      call vcross(rbody,temp1,temp2)              ! { rbody} X{ I}{ rnorm}
c
      t_gg(1)=ggmul*temp2(1)               ! result in body coordinates
      t_gg(2)=ggmul*temp2(2)
      t_gg(3)=ggmul*temp2(3)
c
c *** Aerodynamic drag/torque calculatons
c
```

```
      call aerocomp(v,amat,f_aero_b,t_aero_b)      ! Inertial velocity and attitude are inputs
c                                                  ! Body forces and torques (geometric origin)
c                                                  ! are outputs
c
      call vcross(rcm,f_aero_b,temp1)              ! transfering drag force from geometric
      t_aero_b(1)=t_aero_b(1)+temp1(1)             ! origin to center of mass and adding couple
      t_aero_b(2)=t_aero_b(2)+temp1(2)
      t_aero_b(3)=t_aero_b(3)+temp1(3)
c
c *** Control System Torques ***
c
      t_cont(1)=-(a0*phi(1)+a1*omega(1))   ! result in body coordinates
      t_cont(2)=-(b0*phi(2)+b1*omega(2))
      t_cont(3)=-(c0*phi(3)+c1*omega(3))
c
c *** orbital dynamics (Orbit State Derivatives) ***
c
      call matvec(amat_t,f_aero_b,f_aero)    ! transform aerodynamic drag vector
c                                            ! from body to inertial coordinates
c
      vdot(1)=(1.0D0/xmass)*(f_grav(1)+f_aero(1))    !result in inertial coordinates
      vdot(2)=(1.0D0/xmass)*(f_grav(2)+f_aero(2))
      vdot(3)=(1.0D0/xmass)*(f_grav(3)+f_aero(3))
c
      rdot(1)=v(1)                                   !result in inertial coordinates
      rdot(2)=v(2)
      rdot(3)=v(3)
c
c *** Attitude dynamics
c
      call matvec(xmoi,omega,temp1)
      call vcross(omega,temp1,temp2)
      t_tot(1)=t_GG(1)+t_aero_b(1)+t_cont(1)-temp2(1) !result in body coordinates
      t_tot(2)=t_GG(2)+t_aero_b(2)+t_cont(2)-temp2(2)
      t_tot(3)=t_GG(3)+t_aero_b(3)+t_cont(3)-temp2(3)
      call matvec(xmoi_inv,t_tot,omegadot)           !result in body coordinates
c
c     Converting body rates to Euler angle derivatives
```

```
c
c     (Rotation sequence precludes 90deg angle about y-axis)
c
      cmat(1,1)=1.0D0
      cmat(1,2)=dsin(phi(1))*dtan(phi(2))
      cmat(1,3)=dcos(phi(1))*dtan(phi(2))
      cmat(2,1)=0.0D0
      cmat(2,2)=dcos(phi(1))
      cmat(2,3)=-dsin(phi(1))
      cmat(3,1)=0.0
      cmat(3,2)=dsin(phi(1))/dcos(phi(2))
      cmat(3,3)=dcos(phi(1))/dcos(phi(2))
      call matvec(cmat,omega,phidot)        ! result in non-orthogonal Euler-frame coordinates
c
c *** Changing back to mathematical notation from physical notation.
c
      xdot(1)=rdot(1)
      xdot(2)=rdot(2)
      xdot(3)=rdot(3)
      xdot(4)=vdot(1)
      xdot(5)=vdot(2)
      xdot(6)=vdot(3)
c
      xdot(7)=phidot(1)
      xdot(8)=phidot(2)
      xdot(9)=phidot(3)
      xdot(10)=omegadot(1)
      xdot(11)=omegadot(2)
      xdot(12)=omegadot(3)
c
      return
      end
```

REFERENCES

1. ED11 (12–98–02), "ED Lab High-Fidelity Multidiscipline Simulation," January 1998.

2. ED11 (12–98–114), "ED Lab High-Fidelity Multidiscipline Simulation," June 1998.

3. ED11 (12–98–246), "A Recursive Feedback Algorithm for Simulation of Dynamic Systems," December 1998.

4. ED11 (12–98–251), "An Assorted Collection of Analyses that Support the Concept of Recursive Feedback for Simulation of Dynamic Systems," December 1998.

5. ED11 (12–98–255), "Analytical Demonstration of Solution by Recursive Feedback for a Linear System Composed of Internally Inaccessible Subsystems," December 1998.

6. TD55 (00–18), "Analytical Demonstration of Recursive Feedback as a Solution Method for Simulation of Linear Dynamical Systems with Algebraic Loops," August 2000.

7. Ogata, K.: *State Space Analysis of Control Systems*, Prentice Hall, Englewood Cliffs, NJ, pp. 12–15, 1967.

8. Greenwood, D.T.: *Principles of Dynamics*, Prentice Hall, Englewood Cliffs, NJ, 1965.

9. Wertz, J.R.; and Wiley, J.L.: *Spacecraft Mission Analysis and Design*, 3rd ed., Microcosm Press, Torrance, CA, p. 324, 1999.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 2001	Technical Publication

4. TITLE AND SUBTITLE

System Simulation by Recursive Feedback:
Coupling a Set of Stand-Alone Subsystem Simulations

5. FUNDING NUMBERS

6. AUTHORS

D.D. Nixon

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)

George C. Marshall Space Flight Center
Marshall Space Flight Center, AL 35812

8. PERFORMING ORGANIZATION REPORT NUMBER

M–1032

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546–0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA/TP—2001–211331

11. SUPPLEMENTARY NOTES

Prepared for Vehicle and System Development Department, Space Transportation Directorate

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified-Unlimited
Subject Category 59
Standard Distribution

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Conventional construction of digital dynamic system simulations often involves collecting differential equations that model each subsystem, arranging them to a standard form, and obtaining their numerical solution as a single coupled, total-system simultaneous set. Simulation by numerical coupling of independent stand-alone subsimulations is a fundamentally different approach that is attractive because, among other things, the architecture naturally facilitates high fidelity, broad scope, and discipline independence. Recursive feedback is defined and discussed as a candidate approach to multidiscipline dynamic system simulation by numerical coupling of self-contained, single-discipline subsystem simulations. A satellite motion example containing three subsystems (orbit dynamics, attitude dynamics, and aerodynamics) has been defined and constructed using this approach. Conventional solution methods are used in the subsystem simulations. Distributed and centralized implementations of coupling have been considered. Numerical results are evaluated by direct comparison with a standard total-system, simultaneous-solution approach.

14. SUBJECT TERMS

system simulation, coupled subsystems, multidiscipline, high fidelity, recursive, feedback, dynamic

15. NUMBER OF PAGES

196

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

National Aeronautics and
Space Administration
AD33
George C. Marshall Space Flight Center
Marshall Space Flight Center, Alabama
35812